UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

FERNANDO SOLLER MECKING

InCampus: Sistema Móvel para Localização em Ambientes Outdoor e Indoor Aplicado a um Campus Universitário

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Leandro Krug Wives

Porto Alegre
2015

AGRADECIMENTOS

Gostaria de agradecer aos amigos e colegas da universidade por possibilitarem momentos agradáveis mesmo em meio a rotina difícil de um curso superior, obrigado Gurizada e Void F.C.. Agradeço a minha família pelo suporte, apoio e confiança depositada em mim. Agradeço aos meus avós, que infelizmente não estão mais entre nós, pelo carinho e dedicação. Agradeço aos meus amigos por sempre estarem presentes nos momentos bons, assim como nos momento de dificuldade. Gostaria de agradecer especialmente a minha namorada Gabriela que, além de ser o amor da minha vida, me deu todo o apoio e suporte necessários durante o período de realização deste trabalho e sempre me deu forças para seguir em frente.

Não poderia deixar de agradecer ao meu orientador, professor Leandro Krug Wives, por todo apoio na realização deste trabalho e por sempre se mostrar disponível para ajudar.

"Não importa o quão devagar você vá, desde que você não pare."

– Confúcio

RESUMO

O objetivo deste trabalho é o desenvolvimento de uma aplicação para dispositivos móveis que permita ao usuário se localizar em um campus universitário, possibilitando a visualização dos prédios do campus e suas salas em um mapa interativo, além de permitir ao usuário a pesquisa por nome de salas e pessoas e como resultado exibir a sala procurada no mapa. Ao longo do trabalho são explorados temas como mapas interativos, aplicações móveis híbridas e REST API.

Palavras-chave: Localização. Mapa interativo. Aplicação móvel. REST API.

InCampus: Mobile System for Localization
in Outdoor and Indoor Environments Applied to a University Campus

ABSTRACT

The objective of this work is the development of a application for mobile devices that allows the user to locate himself on a university campus, enabling the visualization of the campus buildings and their rooms on an interactive map and allows the user to search by name rooms and people and as a result display the requested room on the map. Throughout the work are explored topics such as interactive maps, hybrid mobile applications and REST API.

Keywords: Localization. Interactive map. Mobile application. REST API.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
JS	JavaScript
OSM	OpenStreetMap
JOSM	Java OpenStreetMap Editor
XML	*Extensible Markup Language*
GPS	*Global Positioning System*
URL	*Uniform Resource Locator*
JSON	JavaScript *Object Notation*
SDK	*Software Development Kit*
MVC	*Model-view-controller*
HTTP	*Hypertext Transfer Protocol*

SUMÁRIO

1 INTRODUÇÃO

A cada ano muitos alunos novos ingressam no ensino superior. Por exemplo, segundo UFRGS (2015), no ano de 2015 foram oferecidas em torno de quatro mil vagas para ingresso somente na graduação da Universidade Federal do Rio Grande do Sul, isso sem contar com os ingressantes nos cursos de especialização e pós-graduação.

Grande parte desses alunos relata dificuldade para se localizar no campus universitário e encontrar suas salas de aulas e salas de professores, tanto é que a própria universidade disponibiliza um aplicativo[1] onde é possível visualizar os prédios da universidade em um mapa virtual. Todavia, a ferramenta disponível não permite visualizar a localização das salas em um determinado prédio e nem buscar professores e encontrar a sua sala.

Cada vez mais os usuários dão preferência a acessar a Web a partir de dispositivos móveis. Segundo Fling (2009), mais de 1.6 bilhões de pessoas possuem um dispositivo móvel com acesso a internet, o que corresponde a 25% da população mundial, e somente 1.1 bilhão de pessoas acessa a Web através de computadores.

Tendo em vista o exposto anteriormente, o objetivo deste trabalho é o desenvolvimento de uma aplicação para dispositivos móveis que possibilite ao usuário se localizar em um campus universitário, permitindo que ele visualize os prédios do campus e as suas salas em um mapa interativo, além de oferecer a opção de o usuário realizar buscas por nome de sala ou de pessoa, visualizando o resultado da busca no mapa.

O trabalho está organizado em sete capítulos. Após a introdução, o segundo capítulo introduzirá os principais conceitos relacionados ao trabalho, que são: dispositivos móveis e suas plataformas, aplicações móveis híbridas e REST API.

No terceiro capítulo será apresentada a base teórica do trabalho com a descrição das ferramentas que foram utilizadas para o seu desenvolvimento. O capítulo está dividido em ferramentas para mapas, para desenvolvimento de aplicações móveis híbridas e, por último, *frameworks* para o desenvolvimento e consumo de REST API.

O projeto da aplicação será apresentado no capitulo quatro, onde será mostrada a modelagem da arquitetura do aplicativo, além das funcionalidades esperadas representas por *user stories* e da modelagem proposta para o armazenamento de dados.

O desenvolvimento do aplicativo será detalhado no capítulo cinco. Começando pela justificativa para ter sido desenvolvida uma aplicação móvel híbrida, por uma visão geral das

[1] UFRGS Mapas (https://play.google.com/store/apps/details?id=br.ufrgs.ufrgsmapas&hl=pt_BR)

tecnologias utilizadas e a metodologia de desenvolvimento utilizada. Logo após será elucidada a forma como ocorreu o desenvolvimento da REST API, como os dados necessários para a aplicação foram obtidos, de que maneira o mapa foi customizado e os componentes utilizados para construção da interface de usuário.

No capítulo seis será apresentado o funcionamento do aplicativo através da descrição de todas as suas telas, com explicações de como realizar cada operação oferecida pela aplicação.

Finalmente, no capítulo sete, será apresentada a conclusão do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

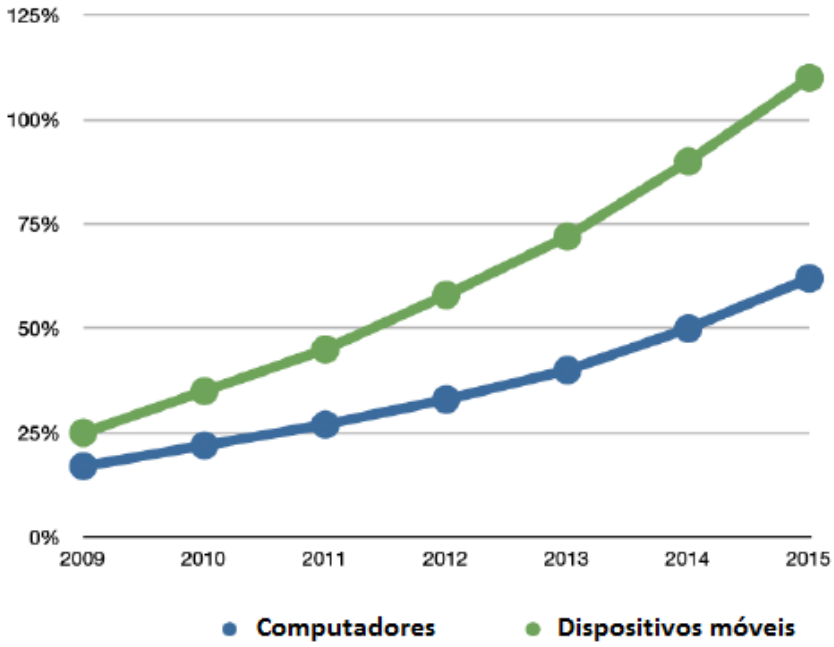
Neste capítulo serão introduzidos conceitos, de suma importância, relacionados ao trabalho, além disso serão descritos aplicativos similares ao proposto pelo trabalho e suas diferenças. Inicialmente será apresentado o conceito de dispositivos móveis, o crescimento do seu uso e o impacto desse fato na criação de serviços baseados em localização, além de discorrer sobre plataformas móveis e sua participação no mercado. Logo após será apresentado o conceito de aplicações móveis híbridas e REST API e por último serão apresentados os trabalhos relacionados.

2.1 Dispositivos móveis

Segundo Firtman (2013), um dispositivo móvel é um aparelho que possui as seguintes características: é portável, é pessoal, está com o usuário quase todo o tempo, é fácil e rápido de se utilizar e possui algum tipo de conexão com a internet.

Cada vez mais os usuários dão preferência a acessar a Web desse tipo de dispositivo. Segundo Fling (2009), nos dias de hoje mais pessoas acessam a Web por dispositivos móveis do que por computadores e ao que tudo indica essa diferença deve crescer ainda mais nos próximos anos, como pode ser visto na Figura 2.1.

Figura 2.1 - Crescimento projetado do acesso a Web por dispositivos móveis e computadores



Fonte: Fling (2009, p.33, tradução nossa).

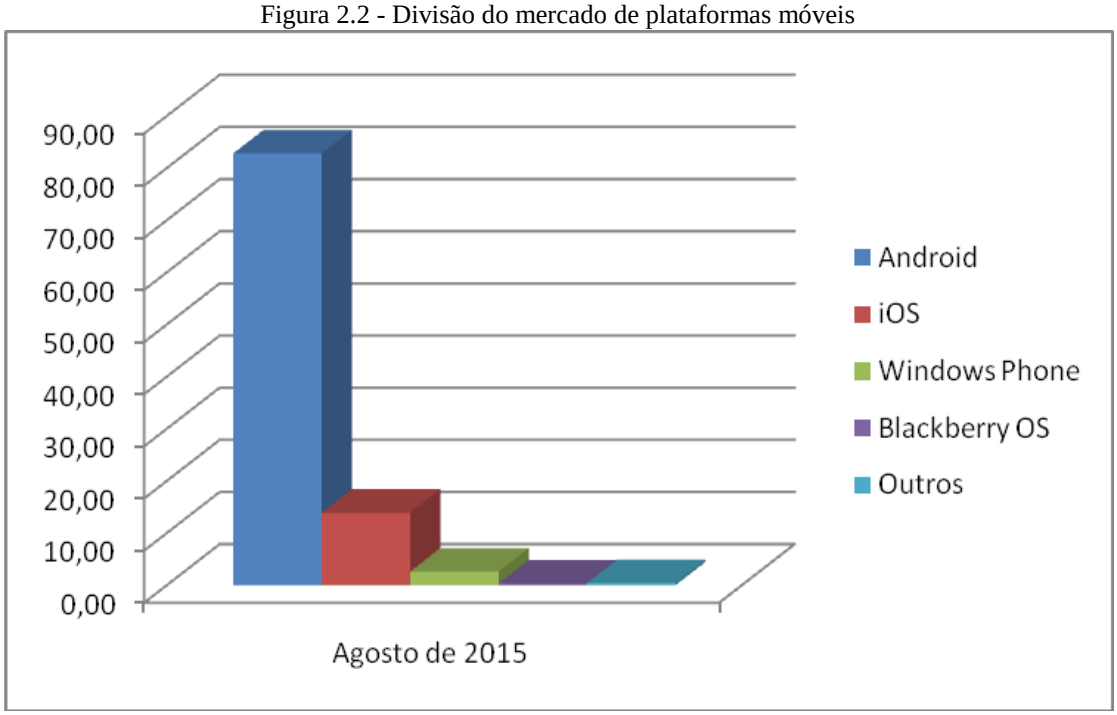
Uma conseqüência direta da maior adoção do acesso a Web por dispositivos móveis é a possibilidade da criação de serviços baseados em localização, que são aplicações que, segundo Fling (2009), se utilizam da localização do usuário como base para obtenção de dados, como por exemplo a aplicação proposta por esse trabalho que se utiliza da localização do usuário para mostrar informações do campus em que ele se encontra. Isso é possível já que a maioria dos dispositivos móveis atuais são equipados com GPS, que segundo Holdener III (2011) é a forma mais precisa de se obter a localização de um dispositivo.

2.1.1 Plataformas móveis

Segundo Fling (2009), o objetivo principal de uma plataforma móvel é prover acesso as funcionalidades do dispositivo móvel, assim como permitir que o usuário execute aplicativos e serviços no dispositivo.

De acordo com dados de IDC (2015), as três principais plataformas móveis disponíveis atualmente são Android, iOS e Windows Phone, que unidas equivalem a 99,3% do mercado de dispositivos móveis, como pode ser observado no gráfico da Figura 2.2.

Figura 2.2 - Divisão do mercado de plataformas móveis



Fonte: Elaborada pelo autor com base em IDC (2015).

2.2 Aplicações móveis híbridas

De acordo com Fling (2009), aplicações móveis nativas ou voltadas para plataforma são aquelas que são desenvolvidas especificamente para dispositivos que utilizam a plataforma em questão. Caso haja o intuito de se criar uma aplicação móvel que funcione nas três principais plataformas do mercado, que são Android, iOS e Windows Phone, como constatado na seção 2.1.1, é necessário desenvolver uma aplicação para cada uma das plataformas.

Outra metodologia para o desenvolvimento de aplicações móveis é a híbrida, que segundo Firtman (2013), é a possibilidade de se desenvolver uma aplicação Web utilizando-se HTML, CSS e JavaScript e criar um pacote nativo com essa aplicação que pode ser distribuído como se fosse uma aplicação nativa, dessa forma é possível desenvolver uma aplicação que funcione em diversas plataformas, incluindo as três principais do mercado. Na seção 3.2.3 será descrito em detalhes o funcionamento de uma ferramenta para criação de aplicações híbridas.

2.3 REST API

Segundo Doglio (2015), REST (sigla de REpresentational State Transfer, Transferência de Estado Representacional, em português), é um estilo arquitetural criado para ajudar a criar e organizar sistemas distribuídos. Um sistema distribuído implementado de forma REST garantirá melhor performance, escalabilidade, simplicidade de interface, facilidade de modificação dos componentes, portabilidade e confiabilidade. Os principais componentes de uma arquitetura REST são os recursos, que são abstrações de entidades que podem ser conceitualizadas, desde um arquivo de imagem até um documento de texto. Os recursos definem o que os serviços farão, o tipo de informação que será transferida e as ações relacionadas.

Ainda, segundo Doglio (ibidem), já que a arquitetura REST foi concebida sobre o protocolo HTTP, são utilizados verbos desse protocolo para identificar qual ação será feita sobre um recurso, as ações possíveis de acordo com o verbo são as seguintes:

a) GET (Obter, em português): Acessa um recurso em modo somente leitura.
b) POST (Postar, em português): Normalmente utilizado para enviar um novo recurso para o servidor, ação de criação.

c) PUT (Colocar, em português): Normalmente utilizado para atualizar um recurso, ação de atualização.
d) DELETE (Excluir, em português): Utilizado para excluir um recurso.
e) HEAD (Cabeçalho, em português): Utilizado para saber se um certo recurso existe sem a necessidade de retornar a sua representação.
f) OPTIONS (Opções, em português): Utilizado para obter uma lista das ações disponíveis sobre um determinado recurso.

Fazendo-se uso dessa arquitetura é possível desenvolver uma aplicação móvel híbrida que se comunique com um banco de dados de forma transparente com a utilização de verbos HTTP através de uma REST API.

2.4 Trabalhos relacionados

Nesta seção serão apresentados aplicativos similares ao proposto pelo trabalho. Serão descritas suas funcionalidades e diferenças com relação ao trabalho proposto.

2.4.1 UFRGS Mapas

O aplicativo UFRGS Mapas possibilita a localização dos alunos nos campi da Universidade Federal do Rio Grande do Sul. No aplicativo é possível pesquisar por nome ou numeração de prédios e visualizá-los em um mapa interativo. É possível também adicionar locais a uma lista de favoritos para rápido acesso. Diferentemente do aplicativo proposto pelo trabalho não é possível visualizar o mapa interno dos prédios com suas salas, pesquisar por salas ou pessoas e o aplicativo não mostra a localização atual do usuário.

Figura 2.3 - UFRGS Mapas

Fonte: Google Play (2015a).

2.4.2 Onde Fica? - UFC

O aplicativo Onde Fica? - UFC é direcionado aos alunos da Universidade Federal do Ceará. O aplicativo permite ao usuário visualizar os prédios do campus com marcadores diferenciados de acordo com o seu tipo e também é possível pesquisar por nome de prédio. Assim como no aplicativo mostrado anteriormente, no Onde Fica - UFC, não é possível visualizar as salas de um prédio e nem pesquisar por nome de salas ou pessoas e a localização atual do usuário não é exibida.

Figura 2.4 - Onde Fica? - UFC

Fonte: Google Play (2015b).

3 TECNOLOGIAS UTILIZADAS

Neste capítulo serão descritas as tecnologias que foram empregadas no desenvolvimento da aplicação proposta pelo trabalho. Primeiramente serão introduzidas as tecnologias que se fizeram necessárias para possibilitar a utilização de um mapa interativo na aplicação, logo após serão elucidadas as tecnologias necessárias para o desenvolvimento da aplicação móvel híbrida e por último quais ferramentas foram utilizadas para criação da REST API.

3.1 Mapas

Nesta seção serão descritas as ferramentas que permitiram a construção do mapa mostrado no aplicativo. Inicialmente será abordada a solução de mapa escolhida, logo após a ferramenta utilizada para a construção dos objetos de mapa necessários para a aplicação e a API que possibilita consultar os dados dos objetos criados, por último é apresentado o *framework* que viabilizou a customização e interatividade do mapa.

3.1.1 OpenStreetMap

Segundo OpenStreetMap (2015a) e OpenStreetMap Foundation (2015a), o OpenStreetMap é um projeto mantido pela OSM Foundation, uma organização sem fins lucrativos, com a finalidade de criar um mapa mundial colaborativo e com dados abertos, isso é, todas as informações contidas no mapa são enviadas por colaboradores voluntários de todas as partes do planeta e essas informações podem ser usadas livremente desde que seja dado crédito ao projeto. Segundo OpenStreetMap Foundation (2015b), existem no momento mais de um milhão de colaboradores.

3.1.2 Java OpenStreetMap Editor

De acordo com OpenStreetMap (2015b) e OpenStreetMap (2015c), Java OpenStretMap Editor (JOSM) é um aplicativo desktop multi-plataforma de código aberto, desenvolvido na linguagem de programação Java, que permite a edição de mapas do OpenStreetMap. O aplicativo foi desenvolvido inicialmente por Immanuel Scholz e

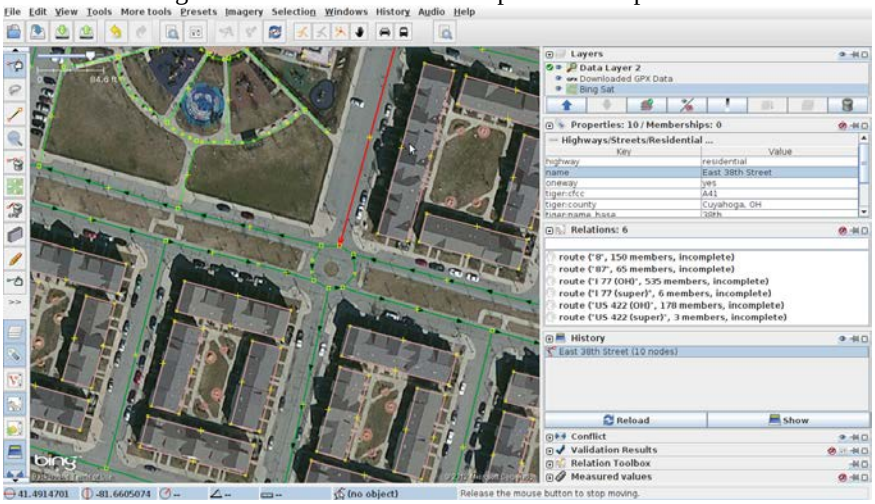
atualmente é mantido por Dirk Stöcker e outros colaboradores. Os principais diferenciais do JOSM é que é possível editar os dados localmente e só enviar para o servidor quando a edição for concluída e ele é extensível, ou seja, é possível instalar *plugins* que provêem novas funcionalidades para o aplicativo.

O aplicativo permite a inserção e edição dos objetos básicos que formam o mapa que, segundo JOSM (2015), são:

a) *Node* (Nodo, em português): Nodo é um ponto com uma posição definida.
b) *Way* (Caminho, em português): Um caminho é uma seqüência de nodos. Apesar do nome, um caminho não representa só um caminho no mundo real. Um caminho no OSM é também usado para representar um rio, as fronteiras de um país ou até mesmo um prédio.
c) *Relation* (Relação, em português): Relação é uma seqüência de outros objetos, por exemplo uma seqüência de caminhos (representando segmentos de uma estrada) e nodos (representando paradas de ônibus) que representam a rota de um ônibus.
d) *Area* (Área, em português): Área é um caminho em que as extremidades estão conectadas, ou seja, o primeiro e o último nodo são o mesmo.

Cada um desses objetos ao ser criado recebe um identificador único do servidor do OSM e para cada um desses objetos básicos é possível definir *tags* (etiquetas, em português) que são pares chave-valor utilizados para descrever um objeto, por exemplo *building=university* adicionado a uma relação informa que ela é um prédio de uma universidade. Na Figura 3.1 podemos ver a interface do *software*.

Figura 3.1 – Interface do Java OpenStreetMap Editor



Fonte: OpenStreetMap (2015b).

3.1.3 Overpass API

A Overpass API é uma API somente leitura que permite através de consultas específicas obter dados de regiões selecionadas do mapa provido pelo OSM. As consultas para a API podem ser realizadas utilizando os formatos XML ou Overpass QL (Overpass *Query Language*) e os resultados por padrão são retornados em formato XML. O resultado retornado contém o conjunto de nodos, caminhos e relações correspondentes a consulta, juntamente com suas coordenadas e etiquetas, com essa informação é possível desenhar no mapa os pontos, linhas e polígonos correspondentes, permitindo assim a visualização dos objetos (OPENSTREETMAP, 2015d) e (OPENSTREETMAP, 2015e).

No site Overpass Turbo[2] é possível executar as consultas desejadas e visualizar o resultado retornado. Nas imagens abaixo é possível ver a comparação de uma mesma consulta utilizando os formatos XML e Overpass QL, além do resultado gráfico gerado por essa consulta.

Figura 3.2 – Comparação de uma mesma consulta utilizando os formatos XML e Overpass QL

XML	Overpass QL
<osm-script> <query type="relation"> <has-kv k="amenity" v="university"/> <bbox-query e="-51.1158305" n="-30.0641225" s="-30.0765966" w="-51.128638"/> </query> <union> <item/> <recurse type="down"/> </union> <print/> </osm-script>	(rel ["amenity"="university"] (-30.0765966,-51.128638, -30.0641225,-51.1158305);); (._;>;); out;

Fonte: Elaborada pelo autor.

[2] Overpass Turbo (http://overpass-turbo.eu/)

Figura 3.3 – Resultado gráfico da consulta a Overpass API

Fonte: Gerada pelo autor no site Overpass Turbo.

3.1.4 Leaflet.js

Leaflet.js é uma biblioteca de código aberto, escrita em JavaScript, que se apresenta como a principal biblioteca para o desenvolvimento de mapas interativos compatíveis com dispositivos móveis e possui a maior parte das funcionalidades que um desenvolvedor necessita com relação a customização de mapas (AGAFONKIN, 2015). Utilizando a Leaflet.js é possível desenhar polígonos sobre o mapa e adicionar marcadores, além disso a biblioteca provê todos os controles de navegação necessários para se explorar o mapa, como zoom e movimentação.

3.2 Aplicação móvel híbrida

Nesta seção serão detalhadas as ferramentas e *frameworks* que possibilitaram o desenvolvimento da aplicação Web e sua transformação em um aplicativo híbrido para dispositivos móveis compatível com as principais plataformas móveis do mercado.

3.2.1 Node.js

Segundo Monteiro (2014), o Node.js é um servidor Web construído com JavaScript e sua proposta principal é assegurar alta performance e escalabilidade para aplicações Web,

além disso, possibilita um menor investimento em hardware em comparação a servidores convencionais, como os de Java, PHP ou .NET, já que suporta centenas de milhares de usuários conectados simultaneamente em um mesmo servidor físico e um servidor convencional suporta em torno de quatro mil.

3.2.2 Node Package Manager (NPM)

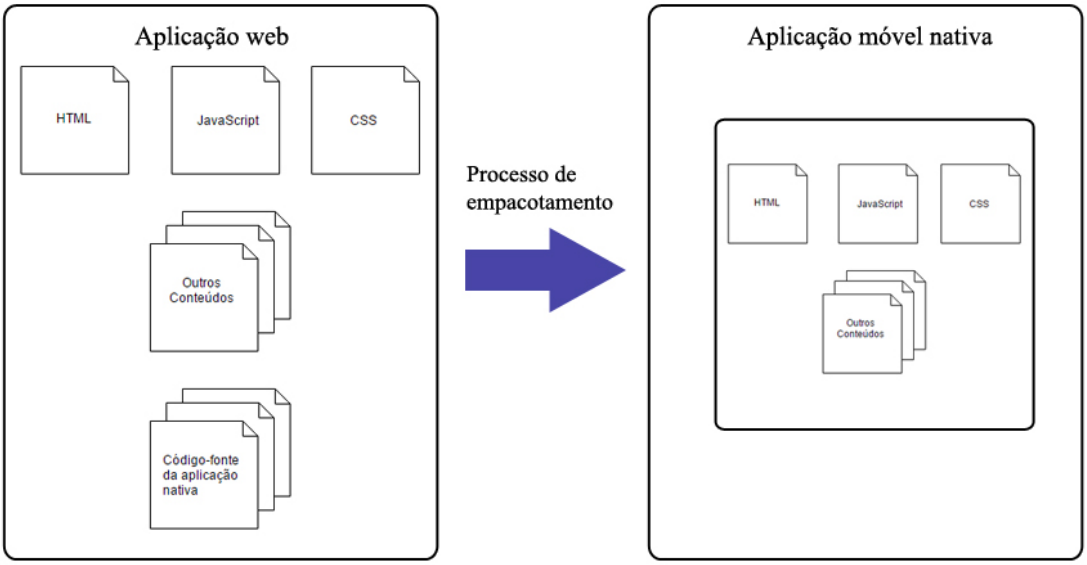
NPM é o gerenciador de pacotes do Node.js, que permite a instalação e atualização de módulos através de uma interface de linha de comando que interage com um repositório online de projetos de código aberto, o repositório contém mais de 76 mil pacotes e novos são adicionados todos os dias (MONTEIRO, 2014).

3.2.3 Apache Cordova

Apache Cordova é um *framework* de código aberto que possibilita a criação de aplicações móveis nativas utilizando HTML, JavaScript e CSS. O aplicativo construído dessa forma é chamado híbrido já que utiliza tecnologias nativas em combinação com tecnologias de aplicações Web. Com ele é possível através do mesmo código-fonte gerar aplicativos para diversas plataformas móveis como Android, iOS e Windows Phone. Isso só é possível por que o Apache Cordova possui aplicações nativas para cada plataforma suportada que servem como um contêiner para que a aplicação Web seja renderizada no dispositivo, além disso provê um conjunto de APIs que permitem a utilização do hardware do aparelho móvel (GPS, câmera, acelerômetro, entre outros) diretamente da aplicação Web renderizada no contêiner nativo, através de chamadas JavaScript (WARGO, 2014).

Ainda, de acordo com Wargo (2014), para que a aplicação possa ser distribuída para diferentes plataformas o *framework* disponibiliza um conjunto de ferramentas para fazer o processo de empacotamento, que se trata de unir a aplicação Web com o contêiner nativo em um pacote que pode ser instalado no dispositivo.

Figura 3.4 - Processo de empacotamento do Apache Cordova

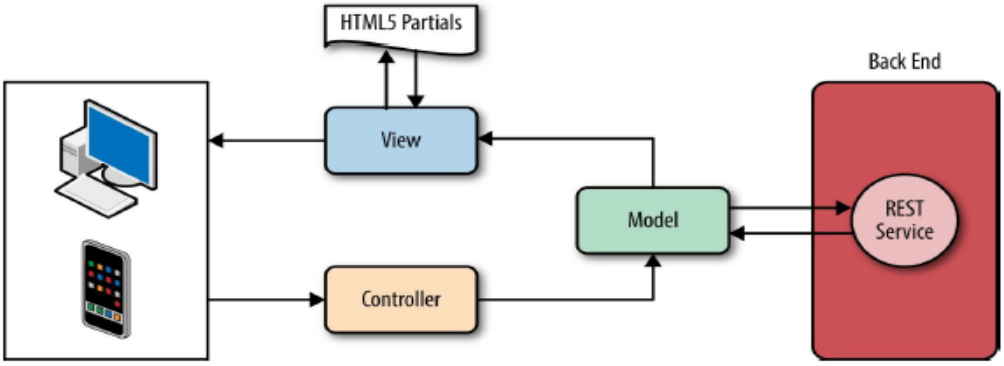


Fonte: Elaborada pelo autor com base em Wargo (2014, p.17).

3.2.4 AngularJS

AngularJS, segundo Willianson (2015) e Seshadri e Green (2014), é um *framework* JavaScript baseado em MVC desenvolvido pelo Google, que torna fácil e rápido o desenvolvimento de aplicações que funcionam bem em qualquer plataforma desktop ou móvel. No momento em que uma aplicação AngularJS é carregada, todo o código-fonte é transferido para o dispositivo desktop ou móvel do usuário e é executado utilizando unicamente o hardware do usuário. Como pode ser observado na Figura 3.5, a aplicação *front-end* desenvolvida utilizando o AngularJS se comunica com uma REST API, onde está localizada toda a lógica de negócios da aplicação e onde é feita a comunicação com o banco de dados da aplicação.

Figura 3.5 - Diagrama de uma aplicação MVC do AngularJS



Fonte: Williamson (2015, p.21)

Existem conceitos básicos da organização do AngularJS que necessitam de elucidação para um correto entendimento do seu funcionamento, esses conceitos são:

a) Controller (Controlador, em português): É responsável por buscar os dados necessários para serem apresentados na interface, pela lógica de apresentação e pela interação do usuário com a aplicação.
b) Directive (Diretiva, em português): São *tags* HTML customizadas que provêem maiores recursos para a camada apresentação da aplicação.
c) Service (Serviço, em português): São funções ou objetos que podem manter um comportamento ou estado através de toda a aplicação. Um *service* pode ser implementado utilizando-se as funções *factory*, *service* ou *provider* do AngularJS.
d) Module (Módulo, em português): É um contêiner onde é possível definir *controllers*, *directives* e *services*. Um módulo pode ser dependente de outros módulos da aplicação.

O AngularJS, segundo Seshadri e Green (2014), oferece funcionalidades que facilitam a implementação de aplicações complexas e de grande porte com facilidade, como o *data-binding* que faz com que a interface gráfica esteja sempre em sincronia com os dados da aplicação, ou seja, quando há uma atualização de dados a interface é automaticamente atualizada, outra funcionalidade central do *framework* é a injeção de dependência que permite que as dependências de um *controller*, *module* ou *service* estejam disponíveis sem a necessidade de se instanciar o objeto. A injeção é feita através de configuração no cabeçalho da função que possui a dependência, na Figura 3.6 podemos observar um módulo que depende dos módulos "lbServices" e "ngCordova".

Figura 3.6 - Exemplo de injeção de dependência no AngularJS

```
angular.module('starter.controllers', ['lbServices','ngCordova'])
```

Fonte: Elaborada pelo autor.

3.2.5 Ionic

Ionic é um SDK de código aberto para *front-end* destinado ao desenvolvimento de aplicações móveis híbridas, ele prove componentes otimizados para plataformas móveis, além de captura de gestos e ferramentas para construir aplicações altamente interativas. Ionic é integrado ao AngularJS já que o utiliza como seu *framework* JavaScript. Possui também uma

ótima integração com o Apache Cordova, permitindo um fácil acesso ao hardware do dispositivo através do uso de bibliotecas próprias. O Ionic possui sua própria interface de linha de comando que é construída sobre o Node.js e permite utilizar as ferramentas providas pelo SDK para desenvolvimento e instalação da aplicação (RAVULAVARU, 2015).

3.3 REST API

Nesta seção serão detalhadas as ferramentas que possibilitaram a criação do servidor e do cliente da REST API.

3.3.1 Loopback

Conforme Strongloop (2015a), Looopback é um *framework* de código aberto construído sobre o Node.js que permite a criação de REST APIs dinâmicas baseadas em modelos, sem a necessidade de codificação. Os modelos e seus relacionamento são configurados em arquivos JSON e através de ferramentas, utilizadas através da interface de linha de comando do *framework*, é possível gerar automaticamente a REST API correspondente. Além disso, o Loopback suporta a maioria dos bancos de dados utilizados comercialmente, como o MySql, Oracle, MongoDB e PostGres.

3.3.2 Loopback AngularJS SDK

De acordo com Strongloop (2015b), é um SDK que pode ser utilizado em conjunto com o *framework* Loopback com o objetivo de gerar automaticamente um *service* para o AngularJS com a representação dos modelos e métodos remotos da REST API. Na Figura 3.6 é mostrado o comando que deve ser utilizado para gerar o *service* à partir do código-fonte do servidor da REST API.

Figura 3.6 - Exemplo de geração do *service* para o AngularJS

```
$ mkdir js
$ lb-ng ../server/server.js js/lb-services.js
```

Fonte: Strongloop (2015b).

4 MODELAGEM E PROJETO DA APLICAÇÃO

Neste capítulo será detalhada a modelagem da arquitetura do aplicativo, assim como o projeto das funcionalidades propostas, através de *user stories*, e do armazenamento dos dados.

4.1 Modelagem da Arquitetura

Nesta seção será relatada a modelagem proposta para a arquitetura do aplicativo móvel. Primeiramente será explicado o padrão arquitetural MVC e como ele foi aplicado ao projeto, posteriormente será descrita a arquitetura geral da aplicação.

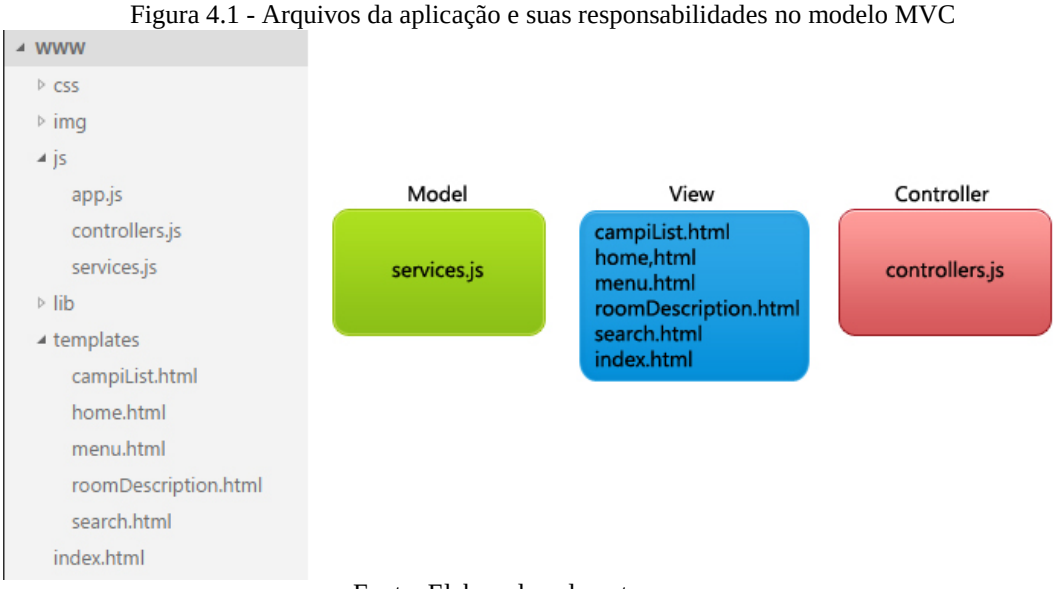
4.1.1 MVC

Segundo Seshadri e Green (2014), o padrão arquitetural MVC (Model-View-Controller) é uma forma de separar unidades lógicas e dividir responsabilidades em uma aplicação. A aplicação é dividida em três partes distintas:

a) *Model* (Modelo, em português): É a parte da aplicação responsável por recuperar dados de um servidor. Qualquer dado dinâmico utilizado na aplicação deve ser derivado do *model*.
b) *View* (Visão, em português): É a interface que o usuário visualiza e com a qual ele interage. A *view* é dinâmica e gerada baseada nos dados providos pelo *model*.
c) *Controller* (Controlador, em português): É camada de apresentação e que possui a lógica de negócios da aplicação. O *controller* é responsável por requisitar dados do *model* e tomar decisões de como esse dado deve ser exibido na *view*.

Na Figura 4.1 é exibido à esquerda os arquivos que compõem a aplicação e à direita como eles são classificados de acordo com a sua responsabilidade na aplicação de acordo com o padrão arquitetural MVC.

Figura 4.1 - Arquivos da aplicação e suas responsabilidades no modelo MVC

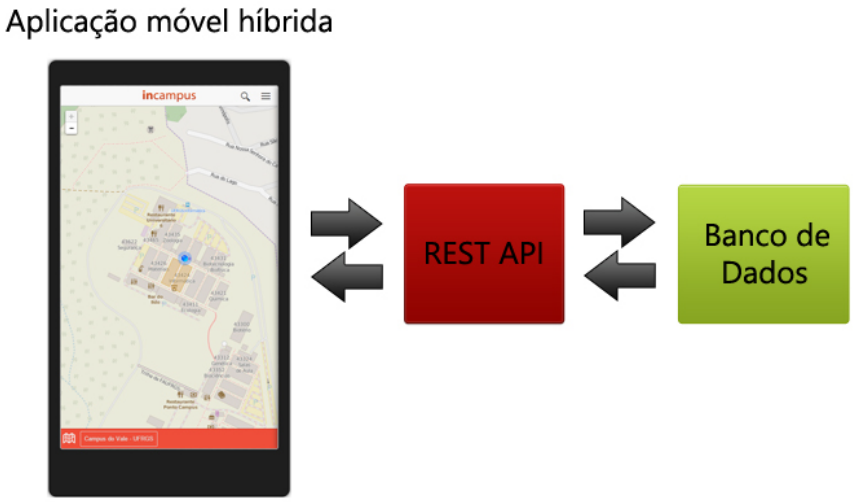


Fonte: Elaborada pelo autor.

4.1.2 Arquitetura Geral

A arquitetura da aplicação segue o preceito utilizado geralmente em aplicativos desenvolvidos utilizando o *framework* AngularJS. A aplicação instalada em um dispositivo móvel se comunica com a REST API através da internet para fazer requisições de dados, as requisições são tratadas pelo servidor da REST API e então os dados necessários para satisfazer a requisição são recuperados do banco de dados e retornados para a aplicação, como pode ser visto na Figura 4.2. A aplicação se comunica unicamente com a REST API e está por sua vez se comunica com a aplicação e com o banco de dados.

Figura 4.2 - Arquitetura geral da aplicação



Fonte: Elaborada pelo autor.

4.2 Modelagem do Aplicativo

Nesta seção serão descritas as funcionalidades requeridas pelo aplicativo, por meio de *user stories*, ademais será detalhada a modelagem do armazenamento de dados.

4.2.1 User Stories

Um dos passos necessários para o desenvolvimento de uma aplicação é definir quais as funcionalidades estarão presentes no produto. A definição das funcionalidades da aplicação proposta por este trabalho foi feita através de *user stories* (histórias de usuário, em português).

Segundo Cohn (2004), uma *user story* descreve uma funcionalidade que será valiosa para um usuário de um *software* e ainda, segundo Stellman e Greene (2015), *user story* é uma maneira de expressar uma necessidade específica que um usuário possui, é geralmente escrita com poucas frases e sem a utilização de termos técnicos.

O formato de escrita de *user stories* utilizado por este trabalho segue o modelo proposto por Cohn (2008): Como um <tipo de usuário>, eu quero <objetivo> para <razão>.

Figura 4.3 - *User stories* da aplicação

User Story 1
Como um Usuário, **eu quero** ver a minha localização no mapa, caso eu esteja em um campus **para** saber onde eu estou em relação aos prédios

User Story 2
Como um Usuário, **eu quero** que ao entrar no aplicativo o mapa mostre o campus em que eu estou, caso eu esteja em um campus **para** rapidamente visualizar os prédios deste campus

User Story 3
Como um Usuário, **eu quero** que ao entrar no aplicativo seja mostrada a lista de seleção de campus, caso eu não esteja em um campus **para** que eu possa escolher qual campus desejo visualizar

User Story 4
Como um Usuário, **eu quero** a qualquer momento poder selecionar o campus que desejo visualizar **para** ver os prédios desse campus e as informações

User Story 5
Como um Usuário, **eu quero** poder ver o mapa interno de um prédio **para** saber a localização das suas salas

User Story 6
Como um Usuário, **eu quero** poder escolher de qual andar do prédio desejo ver o mapa interno **para** poder visualizar as salas dos diferentes andares

User Story 7
Como um Usuário, **eu quero** que as salas no mapa interno do prédio sejam identificadas pelo seu número **para** que eu possa me localizar facilmente sem precisar ver os detalhes da sala

User Story 8
Como um Usuário, **eu quero** poder ver as informações de cada sala **para** saber as características da sala e quais pessoas eu posso encontrar nela

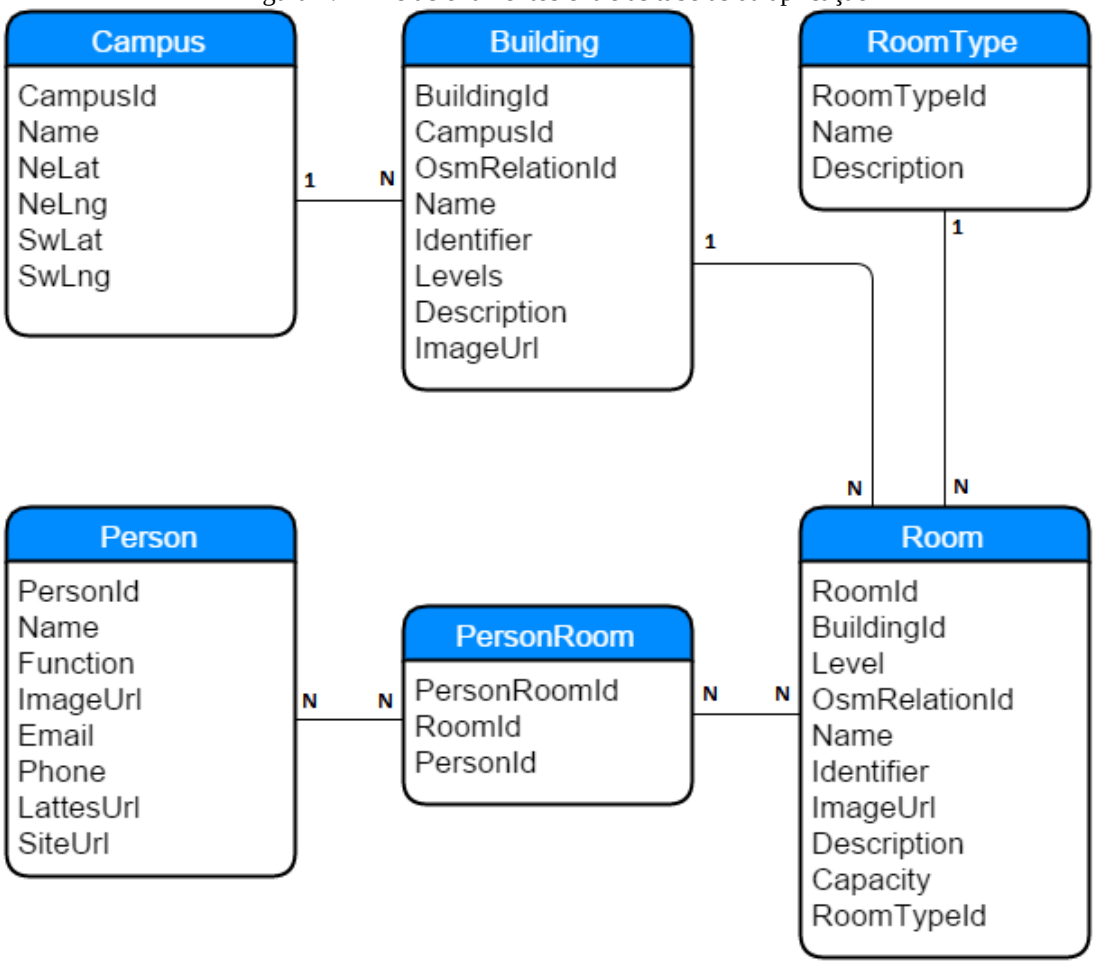
User Story 9
Como um Usuário, **eu quero** poder pesquisar por nome da sala ou de uma pessoa **para** ver a localização no mapa da sala ou pessoa buscada

Fonte: Elaborada pelo autor.

4.2.2 Armazenamento de Dados

Para armazenar os dados da aplicação foi utilizado o MySQL, que segundo Dyer (2008) é o sistema de gerenciamento de banco de dados de código aberto mais popular disponível e mesmo sendo gratuito é muito confiável e rápido. No MySQL os dados são organizados em tabelas e dado seu caráter relacional existe a possibilidade de se criar relações entre as tabelas. Abaixo será apresentado o modelo de relacionamento das tabelas, assim como a descrição de cada uma das tabelas utilizadas para armazenar os dados da aplicação.

Figura 4.4 – Relacionamentos entre as tabelas da aplicação



Fonte: Elaborada pelo autor.

Tabela 4.1 – Descrição das tabelas da aplicação

Nome da tabela	Descrição
Campus	Armazena informações referentes aos campus cadastrados na aplicação
Building	Armazena informações referentes aos prédios cadastrados na aplicação
Room	Armazena informações referentes as salas cadastradas na aplicação
Person	Armazena informações referentes as pessoas cadastradas na aplicação
PersonRoom	Armazena informações referentes ao relacionamento entre salas e pessoas cadastradas na aplicação
RoomType	Armazena informações referentes aos tipos de sala cadastrados na aplicação

Fonte: Elaborada pelo autor.

Tabela 4.2 – Descrição da tabela "Campus"

Nome do campo	Descrição
CampusId	Identificador único do campus
Name	Nome do campus
NeLat	Latitude nordeste do campus

NeLng	Longitute nordeste do campus
SwLat	Latitude sudoeste do campus
SwLng	Longitude sudoeste do campus

Fonte: Elaborada pelo autor.

Tabela 4.3 – Descrição da tabela "Building"

Nome do campo	Descrição
BuildingId	Identificador único do prédio
CampusId	Identificador único do campus ao qual o prédio pertence
OsmRelationId	Identificador único da relação do prédio no OSM
Name	Nome do prédio
Identifier	Número identificador do prédio
Levels	Número de andares do prédio
Description	Descrição do prédio
ImageUrl	URL da imagem do prédio

Fonte: Elaborada pelo autor.

Tabela 4.4 – Descrição da tabela "Room"

Nome do campo	Descrição
RoomId	Identificador único da sala
BuildingId	Identificador único do prédio ao qual a sala pertence
Level	Andar da sala
OsmRelationId	Identificador único da relação da sala no OSM
Name	Nome da sala
Identifier	Número identificador da sala
ImageUrl	URL da imagem da sala
Description	Descrição da sala
Capacity	Capacidade da sala
RoomTypeId	Identificador único do tipo da sala

Fonte: Elaborada pelo autor.

Tabela 4.5 – Descrição da tabela "Person"

Nome do campo	Descrição
PersonId	Identificador único da pessoa
Name	Nome da pessoa
Function	Função da pessoa
ImageUrl	URL da imagem da pessoa
Email	Email da pessoa
Phone	Telefone da pessoa
LattesUrl	URL do cúrriculo Lattes da pessoa
SiteUrl	URL do site da pessoa

Fonte: Elaborada pelo autor.

Tabela 4.6 – Descrição da tabela "PersonRoom"

Nome do campo	Descrição
PersonRoomId	Identificador único do relacionamento entre pessoa e sala
RoomId	Identificador único da sala
PersonId	Identificador único da pessoa

Fonte: Elaborada pelo autor.

Tabela 4.7 – Descrição da tabela "RoomType"

Nome do campo	Descrição
RoomTypeId	Identificador único do tipo de sala
Name	Nome do tipo de sala
Description	Descrição do tipo de sala

Fonte: Elaborada pelo autor.

5 DESENVOLVIMENTO DA APLICAÇÃO

Neste capítulo será exposta a motivação para a escolha do tipo de aplicação desenvolvida, a visão geral das tecnologias utilizadas, a metodologia de desenvolvimento e a explicação de como foi realizado o desenvolvimento das principais funcionalidades do aplicativo.

5.1 Escolha da aplicação híbrida

A aplicação móvel proposta pelo trabalho foi desenvolvida de forma híbrida para possibilitar a compatibilidade com as principais plataformas móveis do mercado. Dessa forma com base na mesma aplicação Web é possível gerar pacotes para instalação do aplicativo nas plataformas Android, iOS e Windows Phone, que são as principais do mercado.

5.2 Visão geral das tecnologias utilizadas

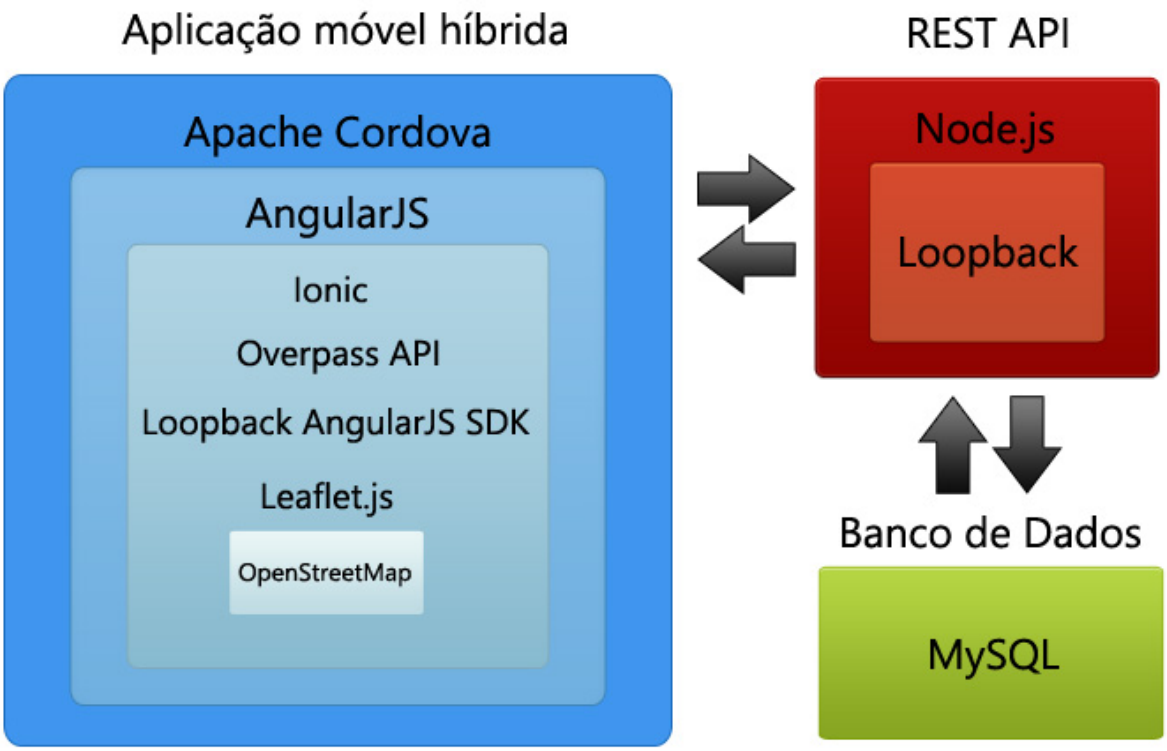
O objetivo desta seção é apresentar uma visão geral das tecnologias utilizadas para desenvolver a aplicação proposta pelo trabalho. No diagrama da Figura 5.1 são mostradas todas as tecnologias que foram utilizadas e em qual contexto.

Para implementar a aplicação móvel híbrida foi utilizado primeiramente o Apache Cordova para que a aplicação Web desenvolvida fosse transformada em uma aplicação móvel. Para desenvolver a aplicação Web foi utilizado o AngularJS como *framework* para JavaScript e por meio dele foram usadas as outras tecnologias que compõem a aplicação, como o Ionic para componentes de *front-end*, a Overpass API para consulta de dados geográficos, o Loopback AngularJS SDK para consulta à REST API e o Leaflet.js que interage com a solução de mapa OpenStreetMap.

Para implementar a REST API da aplicação foi utilizado o *framework* Loopback que funciona sobre o servidor Node.js e acessa o banco de dados MySQL.

Os detalhes da utilização das principais tecnologias da aplicação serão mostrados nas próximas seções.

Figura 5.1 - Visão geral das tecnologias utilizadas



Fonte: Elaborada pelo autor.

5.3 Metodologia de desenvolvimento

Visto que o projeto da aplicação não gerou uma documentação extensiva e poderiam ser necessárias mudanças a qualquer momento, as funcionalidades foram propostas em *user stories* e o desenvolvimento foi iterativo e focado na entrega das funcionalidades a metodologia escolhida para o desenvolvimento foi a Ágil. Segundo Stelmann e Green (2015), o método Ágil possibilita a entrega de uma parte funcional da aplicação a cada iteração de desenvolvimento e mudanças nas funcionalidades a qualquer momento do desenvolvimento, em cada iteração de desenvolvimento são escolhidas *user stories* para serem desenvolvidas e entregues ao fim de um ciclo.

Na Tabela 5.1 são mostradas as iterações de desenvolvimento do aplicativo proposto e quais *user stories* foram desenvolvidas e entregues ao final de cada uma delas.

Tabela 5.1 – Iterações de desenvolvimento

Iteração	Descrição
1	Implementação das *user stories* 1 e 2
2	Implementação das *user stories* 3 e 4
3	Implementação da *user story* 5
4	Implementação das *user stories* 6 e 7
5	Implementação das *user story* 8
6	Implementação das *user story* 9

Fonte: Elaborada pelo autor.

5.4 Desenvolvimento da REST API

A REST API da aplicação foi gerada utilizando-se o *framework* Loopback, que permite a criação de uma REST API completa somente através da definição dos modelos de dados desejados, sem necessidade de codificação. Os modelos de dados são definidos através da criação de um arquivo no formato JSON para cada entidade, que contém as propriedades, os campos e as relações do modelo. Como exemplo, na Figura 5.2 é mostrada a definição do modelo de dados da entidade "Person".

Figura 5.2 – Definição do modelo de dados da entidade "Person"

```
{
  "name": "Person",
  "base": "PersistedModel",
  "idInjection": false,
  "options": {
    "validateUpsert": true
  },
  "properties": {
    "PersonId": {
      "type": "number",
    "generated": true,
    "id": true,
    "required": true
    },
    "Name": {
      "type": "string",
      "required": true
    },
    "Function": {
      "type": "string"
    },
  "ImageUrl": {
      "type": "string"
    },
    "Email": {
      "type": "string"
    },
    "Phone": {
      "type": "string"
    },
    "LattesUrl": {
      "type": "string"
    },
    "SiteUrl": {
      "type": "string"
    }
  },
  "validations": [],
  "relations": {
  "PeopleRooms": {
      "type": "hasAndBelongsToMany",
      "model": "Room",
      "foreignKey": ""
    }
  },
  "acls": [],
  "methods": {}
}
```

Fonte: Elaborada pelo autor.

Em seguida a definição dos modelos é necessário criar as tabelas no banco de dados que será acessado pela REST API, no caso desse trabalho o MySQL, cada modelo corresponde a uma tabela que possui os mesmos campos no banco de dados. O *framework* Loopback permitiu a criação automatizada das tabelas através da execução do *script* que é mostrado na Figura 5.3.

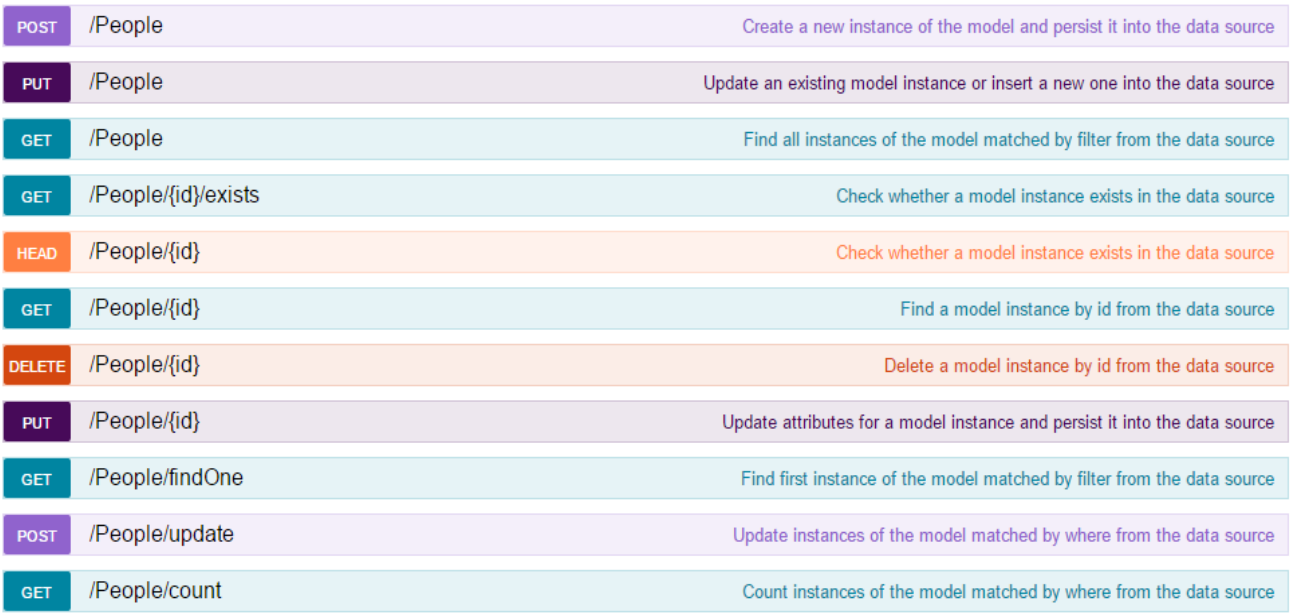
Figura 5.3 – *Script* para criação automática de tabelas no banco de dados

```
1 var path = require('path');
2 var app = require(path.resolve(__dirname, '../server'));
3 var ds = app.datasources.incampusDB;
4
5 ds.automigrate(['Campus', 'Building', 'Room', 'RoomType', 'Person'], function (err) {
6     if (err) throw err;
7 });
```

Fonte: Elaborada pelo autor.

Após a definição dos modelos e a criação das tabelas é possível utilizar a REST API para fazer operações de consulta, inserção, atualização e remoção de dados através do uso de verbos HTTP ou através do LoopBack AngularJS SDK. Com o uso do da ferramenta StrongLoop API Explorer[3] é possível visualizar a REST API gerada, na Figura 5.4 podemos ver os métodos gerados para a entidade "Person".

Figura 5.4 – Visualização dos métodos da REST API gerados para a entidade "Person"



Fonte: Gerada pelo autor utilizando a ferramenta StrongLoop API Explorer.

[3] Strongloop API Explorer (https://docs.strongloop.com/display/public/LB/Use+API+Explorer)

5.5 Obtenção de Dados

Para implementar as funcionalidades propostas para a aplicação foram necessárias diferentes fontes de dados, nas subseções seguintes serão expostas as fontes utilizadas e de que forma os dados foram obtidos de cada uma delas.

5.5.1 Overpass API

A Overpass API foi utilizada para obter dados geográficos dos nodos e relações que compõem os prédios e suas salas, assim como as etiquetas que os descrevem. Os dados obtidos da Overpass API possibilitaram que as plantas dos prédios fossem desenhadas sobre o mapa. Foram utilizadas duas consultas no formato Overpass QL para fazer requisições para o servidor da API, a consulta mostrada na Figura 5.5 obtêm os dados de todos os prédios de um campus, já a mostrada na Figura 5.6 – Consulta para obter os dados geográficos das salas de um prédio obtêm os dados de todas as salas que estão nos limites de um prédio, que é determinado pelas coordenadas de um polígono.

Figura 5.5 – Consulta para obter os dados geográficos dos prédios de um campus

```
451  var buildingsQuery = $rootScope.overpassApiUrl+'?data=' +
452                       'relation["type"="building"]'+
453                       '(' + $scope.boundsToString(bounds) + ');' +
454                       'relation(r)["type"="level"];way(r:"shell")->.x;'+
455                       '(rel(bw.x);rel(br);node(w.x);.x;);out skel qt;';
```

Fonte: Elaborada pelo autor.

Figura 5.6 – Consulta para obter os dados geográficos das salas de um prédio

```
387  var roomsQuery = $rootScope.overpassApiUrl+'?data='+
388                   '(way(poly:"'+ $scope.latLngsToString(poly.getLatLngs()) +'");'+
389                   '<;);(._;>;);out body qt;';
```

Fonte: Elaborada pelo autor.

5.5.2 REST API

Para consultar o banco de dados da aplicação através da REST API foi utilizado o módulo do AngularJS gerado pelo LoopBack AngularJS SDK, que permite a abstração dos verbos HTTP, através do uso de métodos para consulta, inserção, atualização e remoção disponibilizados por ele. Todas as funções de consulta foram organizadas seguindo o paradigma *factory* do AngularJS, que permite que esses métodos possam ser utilizados em

qualquer parte do código da aplicação, na Figura 5.7 - Trecho de código da definição do factory "DataProvider pode ser visto como foi feita a definição desse *factory* que foi chamado de DataProvider.

Figura 5.7 - Trecho de código da definição do factory "DataProvider"

```
19 .factory('DataProvider', ['$q','Campus', 'Building','Room','RoomType','Person', function($q, Campus, Building, Room, RoomType, Person)
20 {
```

Fonte: Elaborada pelo autor.

A seguir serão mostrados exemplos da utilização do módulo gerado pelo LoopBack AngularJS SDK para consulta de dados. Na Figura 5.8 pode ser visto o trecho de código para consultar os *campi* cadastrados no aplicativo, para tanto foi utilizado o método "find" sobre o modelo "Campus", que faz com que sejam retornadas todas as instâncias desse modelo cadastrados no banco de dados. Na Figura 5.9 é mostrado o trecho de código para consultar informações de uma sala, nesse caso foi utilizado o método "findOne" sobre o modelo "Room" e foi definido um filtro com um identificador da sala, além disso foram incluídos modelos relacionados através do comando "include", sendo assim será retornada uma única instância desse modelo que satisfaz os critérios definidos e juntamente as instâncias dos modelos relacionados "PeopleRooms" e "RoomType", o que permite que para cada sala consultada seja retornado ao mesmo tempo as pessoas que estão alocadas casos existem e o tipo da sala, sem necessidade de outras consultas.

Figura 5.8 - Trecho de código do método para consultar os *campi* cadastrados

```
22          getCampi: function()
23          {
24              var deferred = $q.defer();
25
26              Campus.find(
27                      function (data)
28                      {
29                          deferred.resolve(data);
30                      },
31                      function(err)
32                      {
33                          deferred.reject(err);
34                      }
35              );
36
37              return deferred.promise;
38          },
```

Fonte: Elaborada pelo autor.

Figura 5.9 - Trecho de código do método para consultar informações de uma sala

```
57        getRoomInfo: function(roomRelationId)
58        {
59            var deferred = $q.defer();
60
61            Room.findOne({
62                filter: { where: { OsmRelationId: roomRelationId }, include: ['PeopleRooms', 'RoomType']}} ,
63                function (data)
64                {
65                    deferred.resolve(data);
66                },
67                function(err)
68                {
69                    deferred.reject(err);
70                }
71            );
72
73            return deferred.promise;
74        },
```

Fonte: Elaborada pelo autor.

5.5.3 Coordenadas geográficas do usuário

Para se obter as coordenadas geográficas do usuário, que são utilizadas para exibir o marcador de localização, foi utilizado um *plugin* para o Apache Cordova, que através do acesso ao hardware do dispositivo móvel consegue inferir a localização do usuário e retorna sua latitude e longitude. A API do Apache Cordova é acessada pela aplicação através de um módulo para o AngularJS chamado ngCordova[4]. Na Figura 5.10 é mostrado o trecho de código utilizado para obter a latitude e longitude do dispositivo móvel do usuário.

Figura 5.10 - Trecho de código para obtenção das coordenadas geográficas do usuário

```
586                var posOptions = {timeout: 10000, enableHighAccuracy: false};
587                $cordovaGeolocation
588                    .getCurrentPosition(posOptions)
589                    .then(function (position) {
590                        var userLat  = position.coords.latitude
591                        var userLng = position.coords.longitude
```

Fonte: Elaborada pelo autor.

5.6 Customização do Mapa

Para customizar o mapa foi utilizada a biblioteca Leaflet.js que, como foi dito na subseção 3.1.4, permite desenhar polígonos e marcadores no mapa.

[4] ngCordova (http://ngcordova.com/).

5.6.1 Representação dos prédios e salas

Para representar os prédios e salas sobre o mapa foi utilizada uma funcionalidade do Leaflet.js que permite desenhar um polígono com características determinadas e posicioná-lo nas coordenadas geográficas desejadas.

Na Figura 5.11 é mostrado o trecho de código utilizado para representar um prédio sobre o mapa, primeiro é instanciado um polígono (*polygon*) com tamanho de contorno (*weight*) e cor (*color*) determinadas, logo após é definido um identificador e o polígono é adicionado no mapa, finalmente são definidas as coordenadas geográficas do polígono utilizando-se dados provenientes da Overpass API.

Figura 5.11 - Trecho de código para representação de um prédio sobre o mapa

```
367         var buildingPoly = L.polygon([[]], { weight: 1, color: color });
368         buildingPoly._leaflet_id = buildingId;
369         buildingPoly.addTo($rootScope.map);
370         buildingPoly.setLatLngs(OverpassData.latlng);
```

Fonte: Elaborada pelo autor.

A representação das salas é similar ao do prédio como podemos ver no trecho de código da Figura 5.12, assim como para o prédio é criado um polígono utilizando-se os dados provenientes da Overpass API, porém o polígono não é adicionado diretamente no mapa e sim em uma camada que só será exibida ao se visualizar um prédio.

Figura 5.12 - Trecho de código para representação de uma sala sobre o mapa

```
334             var roomPoly = L.polygon([[]], { weight: 1, color: color });
335             roomPoly.addTo($rootScope.levelsLayerGroup);
336             roomPoly.setLatLngs(OverpassData.roomsLatlng);
```

Fonte: Elaborada pelo autor.

5.6.2 Marcadores

No âmbito de mapas interativos, marcadores são utilizados para demarcar a posição ou mostrar identificadores de pontos de interesse no mapa. Na aplicação proposta os marcadores foram utilizados em duas situações distintas: para mostrar o número das salas e para mostrar a localização do usuário.

Para mostrar o número das salas foi utilizada uma modificação do *plugin* para a Leaflet.js chamado Awesome Markers[5], que foi alterado para mostrar o número da sala no

[5] Awesome Markers (https://github.com/lvoogdt/Leaflet.awesome-markers)

lugar do ícone que é exibido por padrão. Na Figura 5.13 pode ser visto o trecho de código que foi utilizado para incluir o marcador no centro do polígono que delimita a sala.

Figura 5.13 - Trecho de código para adição no mapa de um marcador de sala

```
337             var roomMarkerIcon =
338                 L.AwesomeMarkers.icon({
339                     icon: '',
340                     markerColor: 'orange',
341                     prefix: 'fa',
342                     html: roomNumber
343                 });
344             var roomMarker = new L.marker(roomPoly.getBounds().getCenter(), { icon: roomMarkerIcon });
345             roomMarker.addTo($rootScope.levelsLayerGroup);
```

Fonte: Elaborada pelo autor.

O marcador de localização foi implementado utilizando-se o *plugin* para a Leaflet.js chamado User Marker[6]. Na Figura 5.14 é mostrado o trecho de código que adiciona o marcador ao mapa nas coordenadas geográficas do usuário obtidas pela API do Apache Cordova.

Figura 5.14 - Trecho de código para adição no mapa de um marcador de localização do usuário

```
595                     var userMarker = L.userMarker([userLat, userLng], {pulsing:true, smallIcon:true});
596                     userMarker.addTo($rootScope.map);
```

Fonte: Elaborada pelo autor.

5.7 Interface de Usuário

Para desenvolver o *front-end* da aplicação, isto é, a interface para o usuário, utilizou-se o Ionic, que provê componentes de JavaScript, estilos de CSS e ícones para composição da interface. Na subseção 5.7.1 serão descritos os principais componentes utilizados.

5.7.1 Componentes utilizados

Abaixo serão listados os principais componentes, do Ionic, utilizados na interface da aplicação:

a) Header: Foi utilizado para compor a barra superior da aplicação, onde está localizado o logotipo e os botões de acesso a busca e ao menu lateral.

b) Footer: Foi o componente utilizado para construir a barra inferior da aplicação, onde é exibido o nome do campus ou os andares e o nome do prédio selecionado.

[6] User Marker (https://github.com/heyman/leaflet-usermarker)

c) Buttons: Os botões de ação da aplicação foram construídos utilizando este componente.
d) Radio Button List: Foi empregado na tela de seleção de campus, onde somente uma opção pode ser escolhida por vez e a opção atualmente selecionada é destacada com um ícone à direita e uma cor diferenciada.
e) Cards: Foi utilizado na tela de informações de sala para diagramar os dados de uma maneira que melhore a sua visualização.
f) ionicons: É uma coleção de ícones de onde foram retirados todos os ícones utilizados na aplicação.
g) Side Menu: Foi utilizado para se criar o menu lateral da aplicação.
h) Popup: Permitiu a criação das telas de seleção de campus e informações de sala de forma que elas possam ser abertas sobre o mapa e fechadas sem afetar o estado atual da aplicação.

6 FUNCIONAMENTO DO APLICATIVO

Este capítulo tem como objetivo explicar o funcionamento do aplicativo, para esse fim serão mostradas as telas da aplicação com a descrição detalhada das funcionalidades que cada uma delas possui.

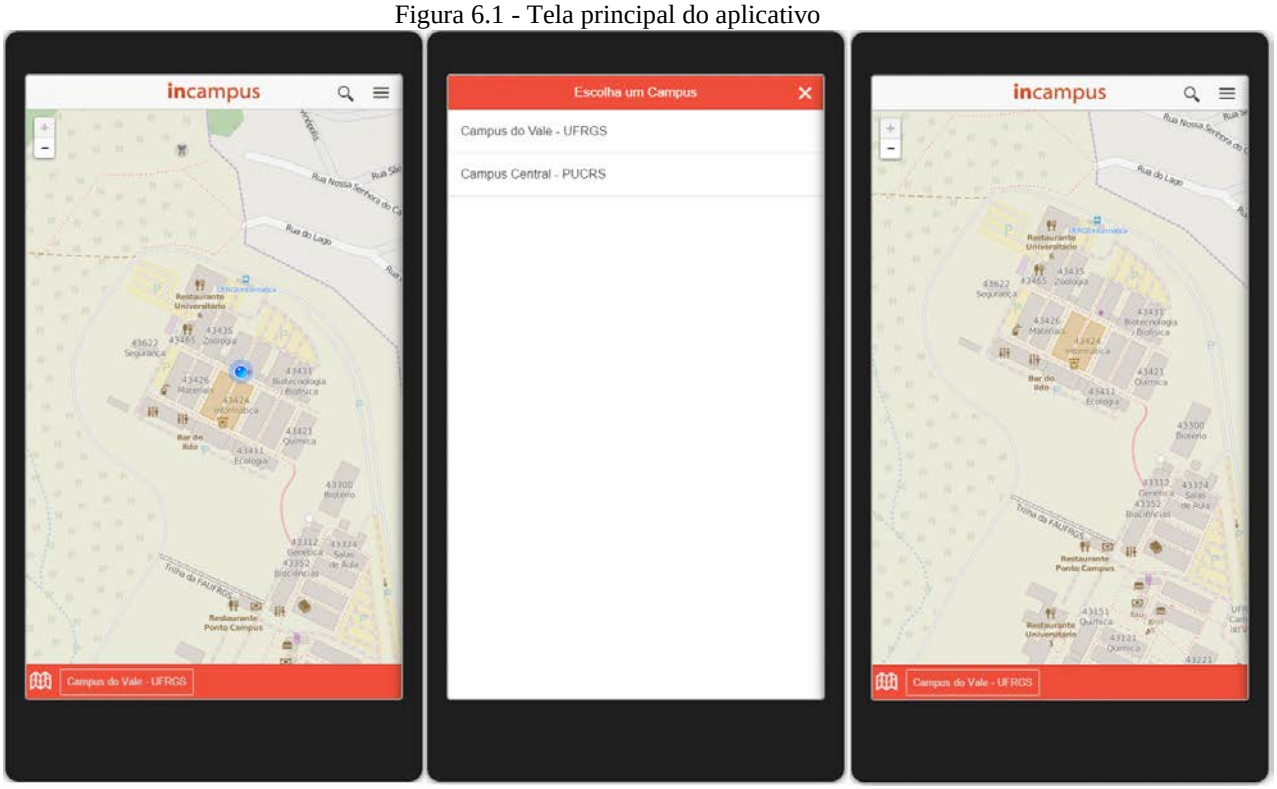
6.1 Tela principal

Ao entrar na aplicação a tela principal é exibida para o usuário. Nesse momento existem duas situações possíveis de acordo com a localização do usuário, o usuário pode estar dentro dos limites de um campus universitário ou não.

Se o usuário estiver dentro dos limites de um dos campus universitários cadastrados no aplicativo, o campus será mostrada no mapa com os prédios que pertencem a este campus destacados em laranja e será exibido o marcador de localização, que é uma circunferência azul, como podemos ver na tela à esquerda da Figura 6.1. Caso contrário, isso é, se o usuário não estiver dentro dos limites de um dos campus universitários cadastrados no aplicativo, a tela de seleção de campus mostrada na tela no centro da Figura 6.1 é exibida e clicando no campus escolhido ele é mostrado no mapa com os seus prédios destacados em laranja, mas sem o marcador de localização do usuário, como podemos ver na tela à direita da Figura 6.1.

Além disso, a partir da tela principal é possível controlar o nível de zoom do mapa através dos botões com os símbolos de mais e de menos, acessar a busca, acessar o menu lateral, fazer a seleção de prédio e fazer a seleção de campus a qualquer momento.

Figura 6.1 - Tela principal do aplicativo



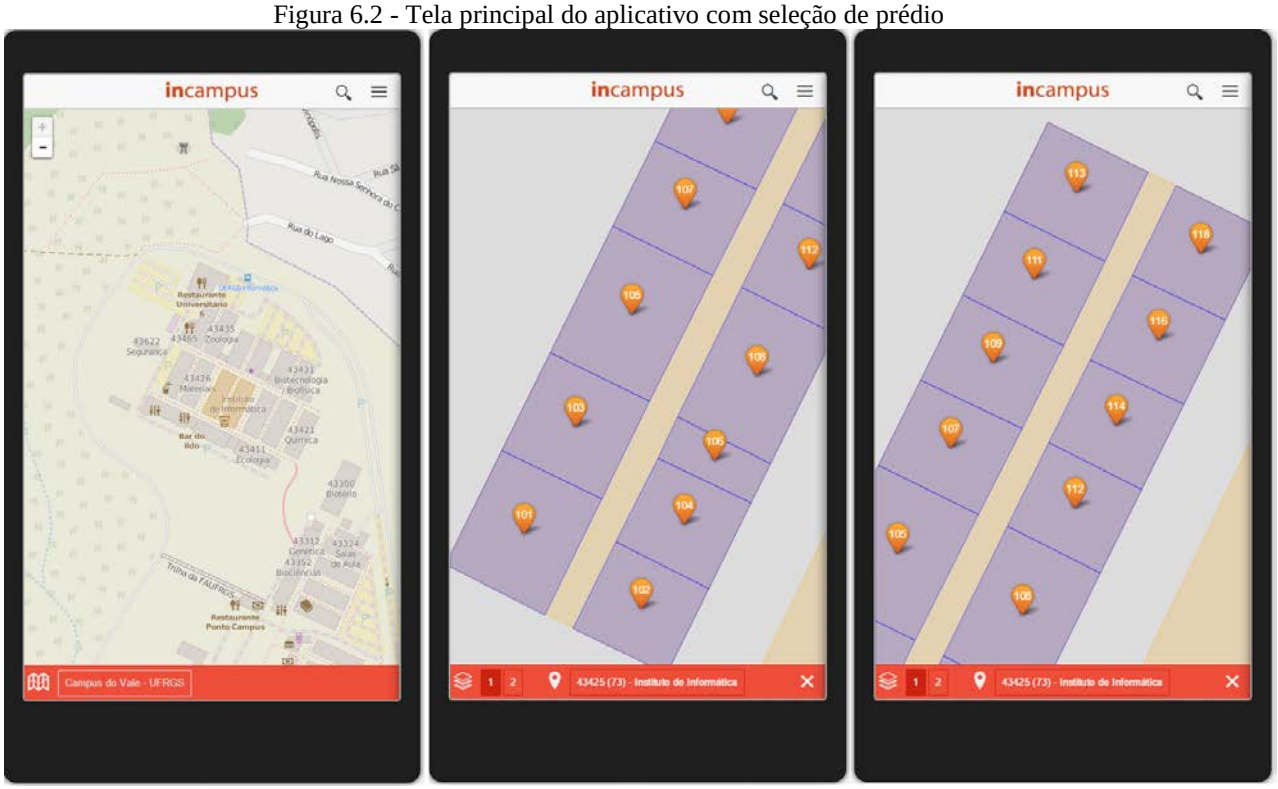
Fonte: Elaborada pelo autor.

6.1.1 Seleção de prédio

O grande diferencial do aplicativo é a possibilidade de visualizar a planta interna dos prédios com a localização das salas que o compõem, para acessar essa funcionalidade o usuário clica no polígono laranja que delimita o prédio, que podemos ver à direita da Figura 6.2 , após isso o aplicativo muda para o modo de visualização de prédio no qual são exibidos na barra inferior os andares que o prédio possui, além do número e nome do prédio e não é possível fazer zoom do mapa, como pode ser visto na tela no centro da Figura 6.2. No modo de visualização de prédio é possível mover o mapa para qualquer um dos lados possibilitando que as salas sejam vistas em sua totalidade, como pode ser observado na diferença de posicionamento entre o mapa na tela do centro da Figura 6.2 na tela à direita da Figura 6.2.

Ao se clicar no ícone em forma de xis no canto direito da barra inferior o modo de visualização de prédio é fechado e o campus atualmente selecionado é mostrado novamente no mapa.

Figura 6.2 - Tela principal do aplicativo com seleção de prédio

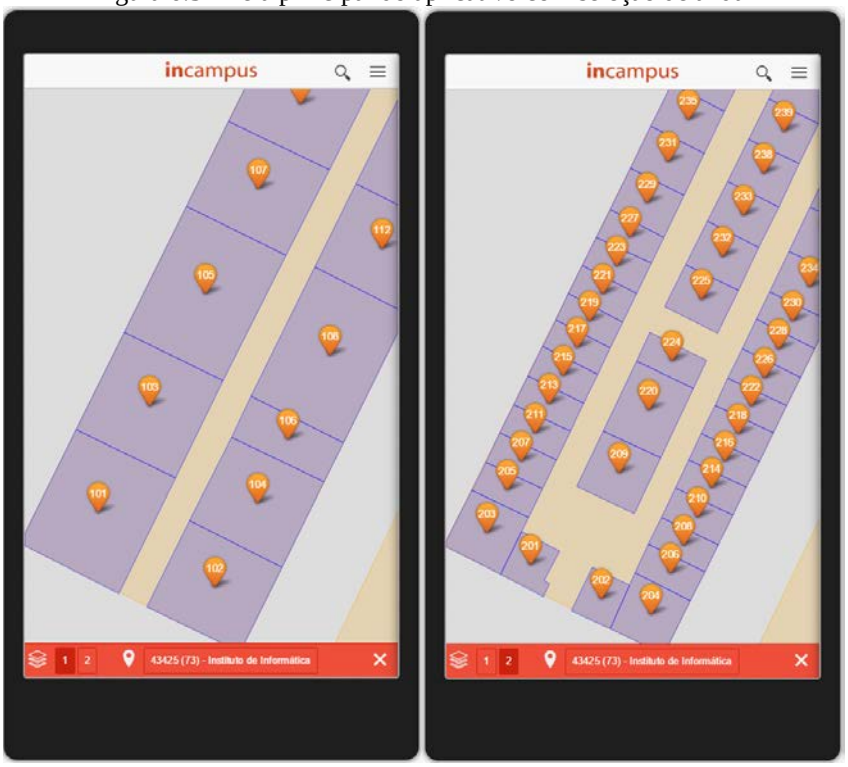


Fonte: Elaborada pelo autor.

6.1.2 Seleção de andar

No modo de visualização de prédio é possível facilmente navegar entre os andares do prédio selecionado ao se clicar no número do andar desejado na barra inferior, o andar selecionado aparece destacado em vermelho escuro. Na tela à esquerda da Figura 6.3 pode ser visto o primeiro andar de um prédio e na tela à direita da Figura 6.3 pode ser visto o segundo andar do mesmo prédio.

Figura 6.3 - Tela principal do aplicativo com seleção de andar

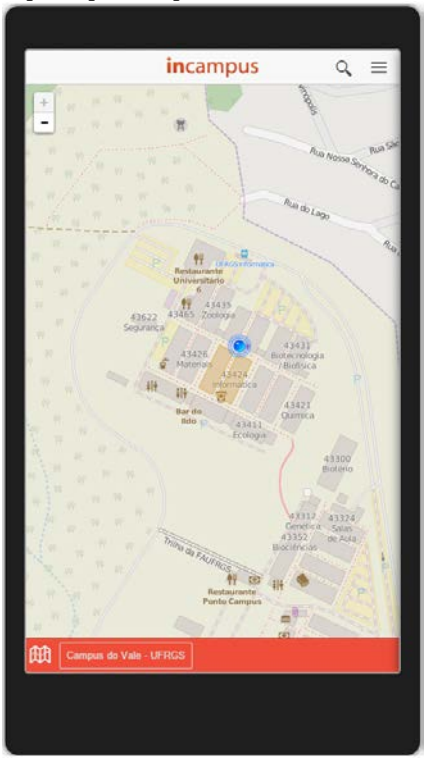


Fonte: Elaborada pelo autor.

6.1.3 Marcador de Localização

Se o usuário estiver dentro dos limites do campus universitário atualmente selecionado no aplicativo ele poderá visualizar facilmente a sua localização em relação aos prédios do campus através do marcador de localização, que é uma circunferência azul como pode ser visto na Figura 6.4, dessa forma se torna muito simples para o usuário saber como chegar ao destino desejado.

Figura 6.4 - Tela principal do aplicativo com marcador de localização

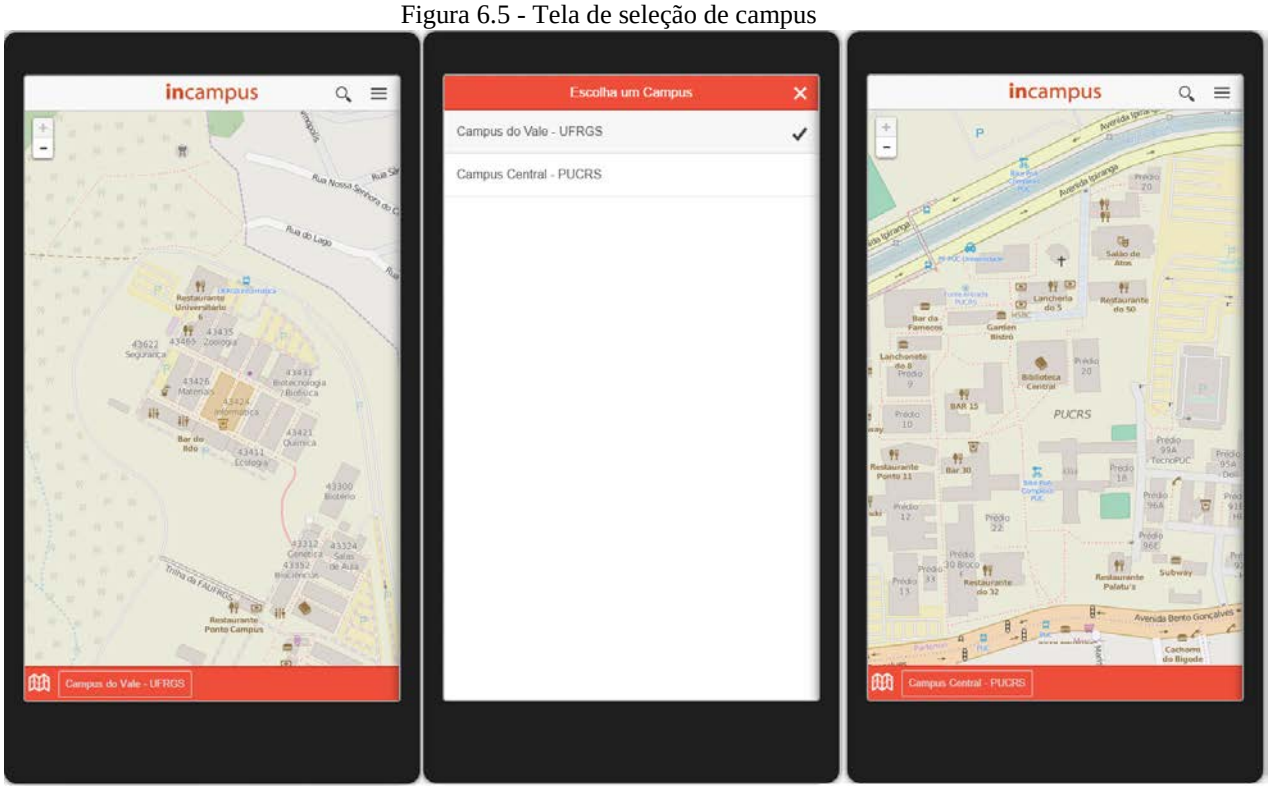


Fonte: Elaborada pelo autor.

6.2 Tela de seleção de campus

Essa tela permite ao usuário selecionar o campus que ele deseja visualizar no mapa, a tela é acessada ao se clicar no nome do campus atualmente selecionado na barra inferior, como pode ser visto à esquerda da Figura 6.5. Ao abrir a tela, que pode ser vista no centro da Figura 6.5, são mostrados os campus cadastrados no aplicativo com um destaque ao campus atualmente selecionado, ao se clicar no nome do campus escolhido a tela se fecha automaticamente e o campus é exibido no mapa, como pode ser visto à direita da Figura 6.5.

Figura 6.5 - Tela de seleção de campus

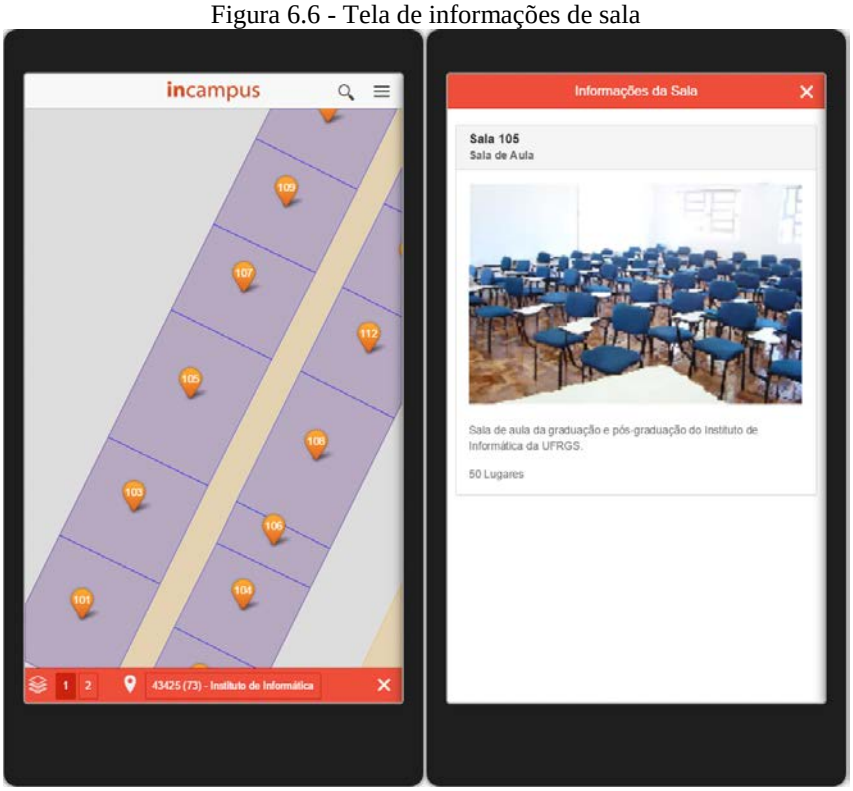


Fonte: Elaborada pelo autor.

6.3 Tela de Informações de Sala

Nessa tela do aplicativo é possível visualizar os detalhes de uma sala, como seu nome, tipo (sala de aula, sala de professor, laboratório, secretaria, etc.), sua foto, sua descrição e o número de pessoas que ela comporta como pode ser visto à direita da Figura 6.6.

Figura 6.6 - Tela de informações de sala

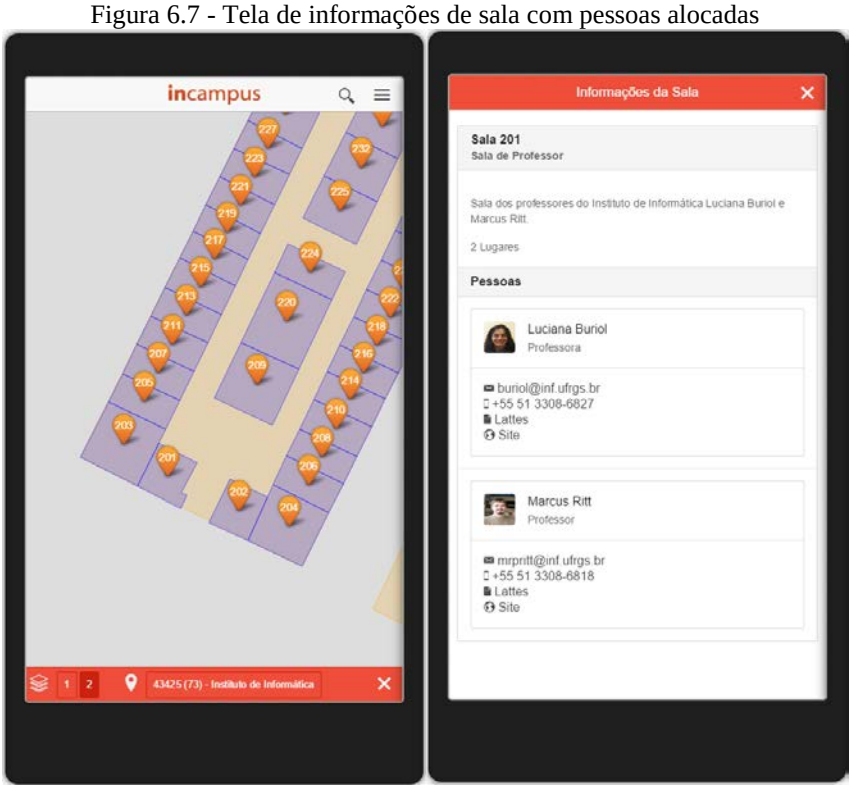


Fonte: Elaborada pelo autor.

Caso existam pessoas alocadas na sala é exibida uma lista com o nome, foto, função, email, telefone, link para o currículo Lattes e link para o site de cada uma das pessoas alocadas na sala como é mostrado à direita da Figura 6.7.

A tela de informações de sala é acessada ao se clicar no polígono azul que delimita a sala ou no marcador laranja que mostra o seu número, como pode ser visto à esquerda da Figura 6.7 e à esquerda da Figura 6.7 .

Figura 6.7 - Tela de informações de sala com pessoas alocadas

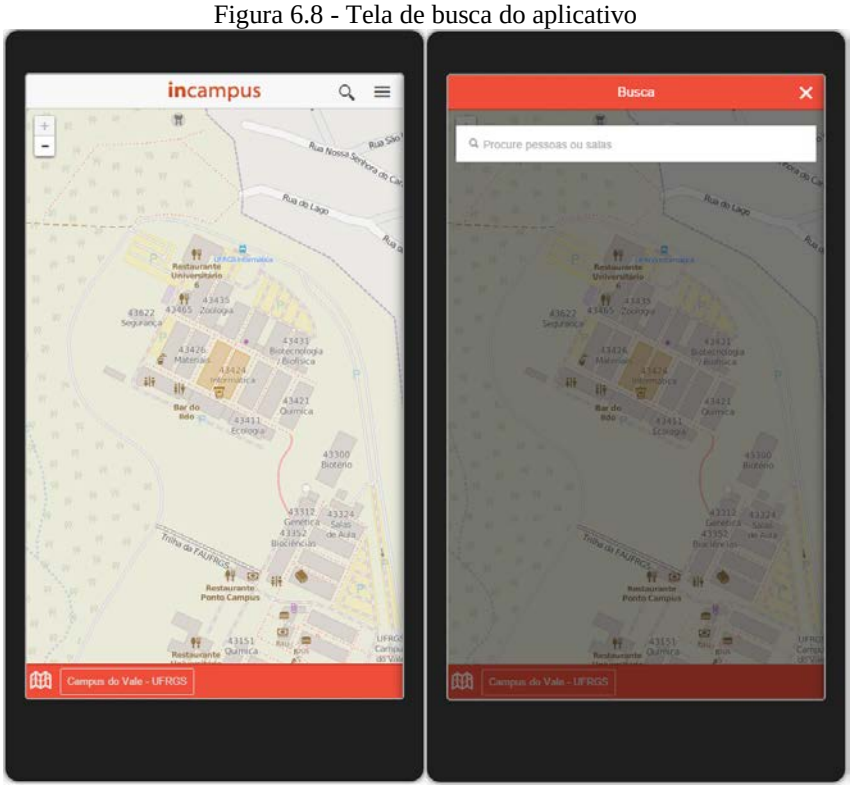


Fonte: Elaborada pelo autor.

6.4 Tela de Busca

Na tela de busca do aplicativo é possível encontrar salas em um campus buscando por nome de sala ou por nome de pessoa, a medida que se digita no campo de busca, que pode ser visto à direita da Figura 6.8, os resultados são filtrados e mostrados. A tela pode ser acessada ao se clicar no ícone de lupa na barra superior do aplicativo, como pode ser visto à esquerda da Figura 6.8.

Figura 6.8 - Tela de busca do aplicativo

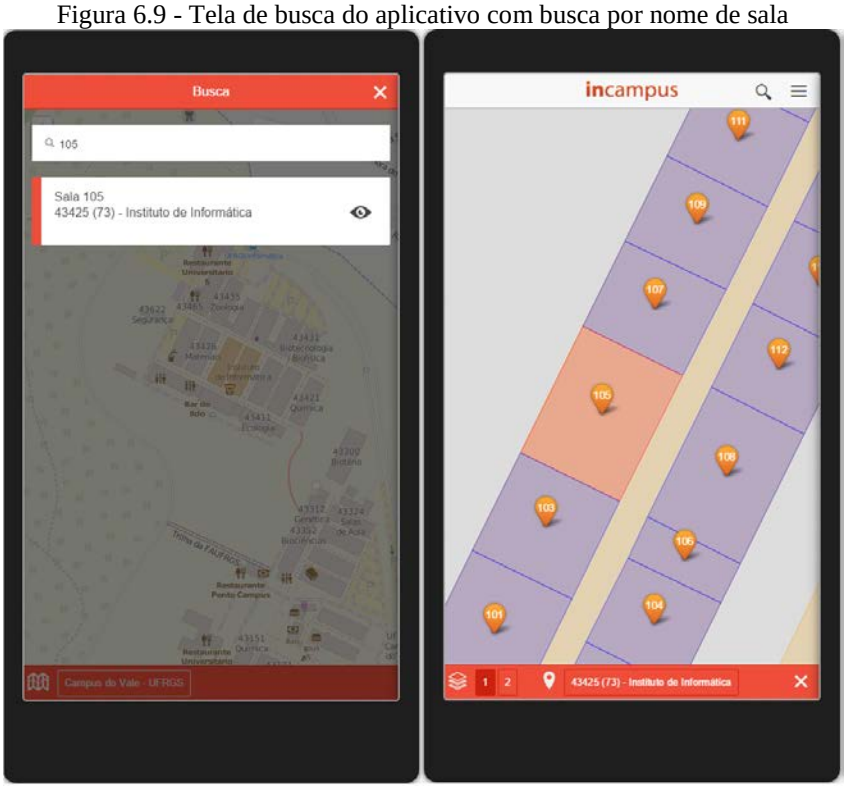


Fonte: Elaborada pelo autor.

6.4.1 Busca por nome de sala

Uma das opções de busca possíveis no aplicativo é pesquisar por nome de sala, o usuário pode digitar qualquer parte do nome da sala e os resultados são filtrados e mostrados, como pode ser visto à esquerda da Figura 6.9 é exibido o nome da sala e a qual prédio ela pertence, além de um ícone de olho que ao ser clicado muda o aplicativo para o modo de visualização de prédio e mostra a sala procurada destacada em vermelho, como mostrado à direita da Figura 6.9.

Figura 6.9 - Tela de busca do aplicativo com busca por nome de sala

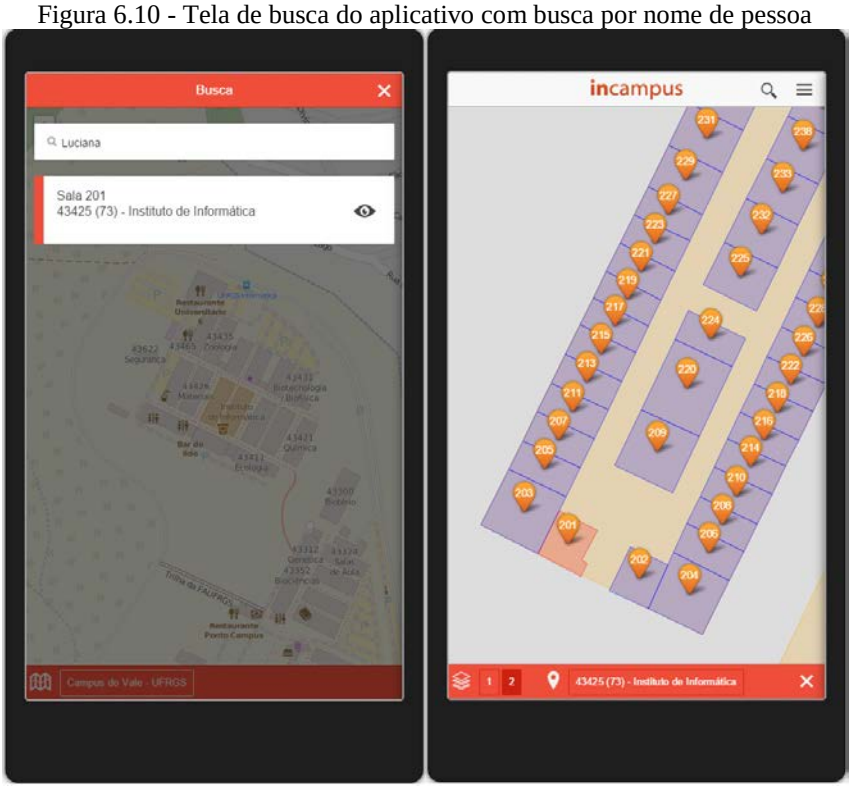


Fonte: Elaborada pelo autor.

6.4.2 Busca por nome de pessoa

Outra opção de busca disponível no aplicativo é pesquisar por nome de pessoa, o usuário pode digitar qualquer parte do nome da pessoa e como resultado são mostradas as salas em que as pessoas encontradas estão alocadas, como pode ser visto à esquerda da Figura 6.10, da mesma forma que na busca por nome de sala ao se clicar no ícone de olho o aplicativo muda para o modo de visualização de prédio e a sala procurada é destacada em vermelho, como pode ser visto à direita da Figura 6.10.

Figura 6.10 - Tela de busca do aplicativo com busca por nome de pessoa

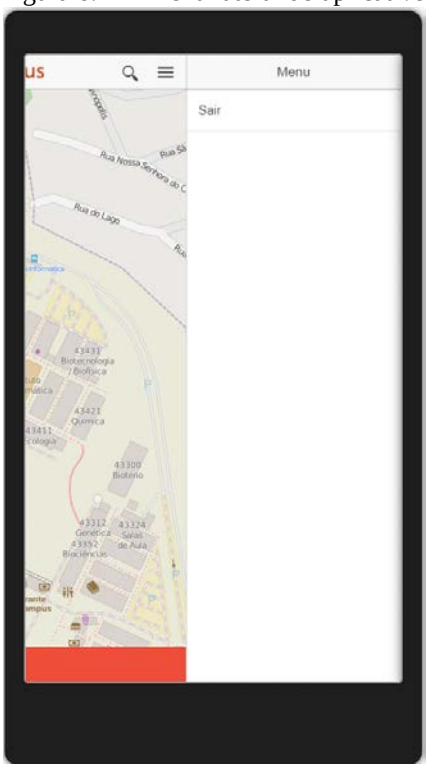


Fonte: Elaborada pelo autor.

6.5 Menu Lateral

Ao se clicar no ícone com três listras paralelas na barra superior do aplicativo é exibido o menu lateral, como pode ser visto na Figura 6.11, que possui uma animação de abertura da direita para a esquerda. Neste menu é possível acessar a opção para sair da aplicação e podem ser incluídas outras opções de acordo com a necessidade.

Figura 6.11 - Menu lateral do aplicativo



Fonte: Elaborada pelo autor.

7 CONCLUSÃO

No transcorrer do trabalho foi demonstrado o crescimento do acesso a Web por dispositivos móveis, que já ultrapassa o acesso por computadores. Além disso, foi exposto o número de alunos de graduação que ingressaram na Universidade Federal do Rio Grande do Sul no corrente ano e o relato da dificuldade de muitos deles de se localizar no campus universitário ainda desconhecido. Nessa conjuntura foi definido o objetivo principal do trabalho como o desenvolvimento de uma aplicação móvel para localização em um campus universitário, que permita ao usuário visualizar os prédios de um campus universitários e suas salas em um mapa interativo, além de possibilitar a busca por nome de sala ou nome de pessoa e como resultado exibir a sala procurada no mapa.

Com o objetivo definido e levando-se em consideração os conceitos relacionados, que demonstram quais são as principais plataformas móveis do mercado foi decidido que a aplicação desenvolvida seria do tipo híbrida, permitindo que a mesma aplicação Web seja distribuída para diferentes plataformas, sem a necessidade do desenvolvimento de múltiplos aplicativos, assim como foi definida a utilização de uma REST API.

Tendo em mente o tipo de aplicativo a ser desenvolvido e as necessidades funcionais da aplicação foram pesquisadas ferramentas que possibilitassem o seu desenvolvimento e cada uma das tecnologias utilizadas foi descrita no trabalho. Ademais, haja vista as funcionalidades propostas e as decisões técnicas efetuadas, foi realizado o projeto arquitetural e a modelagem das funcionalidades e do armazenamento de dados da aplicação.

Ao final do trabalho, como resultado do desenvolvimento, foi obtida uma aplicação híbrida com as funcionalidades propostas, sendo assim o objetivo do trabalho foi plenamente cumprido.

7.1 Trabalhos Futuros

O objetivo do trabalho foi plenamente cumprido, mas durante o seu desenvolvimento surgiram idéias de novas funcionalidades que poderiam agregar valor ao aplicativo. Primeiramente, as opções de busca poderiam ser ampliadas para possibilitar também a consulta por nome de prédio e nome de campus. Outra possibilidade é a integração com um sistema de comentários para que os usuários pudessem compartilhar informações sobre a

infra-estrutura de uma sala ou até mesmo sobre a qualidade de um estabelecimento comercial do campus.

Além disso, poderia ser incluído no aplicativo um sistema de recomendação baseado em localização e no perfil do usuário, ou seja, de acordo com a proximidade do usuário de uma sala ou prédio e das informações contidas em seu perfil o aplicativo faria recomendações contextualizadas. Por exemplo, caso o usuário se aproximasse de uma biblioteca e existisse um livro novo sobre um assunto de seu interesse o aplicativo faria uma notificação, em tempo real, informando o título do livro e sua localização na biblioteca.

REFERÊNCIAS

JOSM. **Objects - The basic building blocks of OSM data and maps.** Disponível em: <https://josm.openstreetmap.de/wiki/Help/Concepts/Object>. Acesso em: Novembro de 2015.

OPENSTREETMAP FOUNDATION. **Main Page.** Disponível em: <http://wiki.osmfoundation.org/wiki/Main_Page>. Acesso em: Novembro de 2015.

OPENSTREETMAP. **About.** Disponível em: <http://www.openstreetmap.org/about>. Acesso em: Novembro de 2015.

OPENSTREETMAP. **JOSM.** Disponível em: <http://wiki.openstreetmap.org/wiki/JOSM>. Acesso em: Novembro de 2015.

OPENSTREETMAP. **Editors.** Disponível em: <http://wiki.openstreetmap.org/wiki/Editors>. Acesso em: Novembro de 2015.

OPENSTREETMAP. **Overpass API.** Disponível em: <http://wiki.openstreetmap.org/wiki/Overpass_API>. Acesso em: Novembro de 2015.

OPENSTREETMAP. **Overpass API/Language Guide.** Disponível em: <http://wiki.openstreetmap.org/wiki/Overpass_API/Language_Guide>. Acesso em: Novembro de 2015.

OPENSTREETMAP FOUNDATION. **About.** Disponível em: <http://wiki.osmfoundation.org/wiki/About>. Acesso em: Novembro de 2015.

WARGO, John M.. **Apache Cordova 3 Programming.** New Jersey: Addison-Wesley, 2014.

COHN, Mike. **User Stories Applied:** for Agile Software Development. Boston: Addison-Wesley, 2004.

STELLMAN, Andrew; GREENE, Jennifer. **Learning Agile.** Sebastopol: O'Reilly, 2015.

COHN, Mike. **Advantages of the "As a user, I want" user story template.** Disponível em: <http://www.mountaingoatsoftware.com/blog/advantages-of-the-as-a-user-i-want-user-story-template >. Acesso em: Novembro de 2015.

DYER, Russel J. T.. **MySQL in a Nutshell.** Sebastopol: O'Reilly, 2008.

MONTEIRO, Fernando. **Learning Single-page Web Application Development.** Birmingham: Packt Publishing, 2014.

RAVULAVARU, Arvind. **Learning Ionic.** Birmingham: Packt Publishing, 2015.

AGAFONKIN, Vladimir. **Leaflet Overview.** Disponível em: <http://www.leafletjs.com>. Acesso em: Novembro de 2015.

WILLIAMSON, Ken. **Learning AngularJS.** Sebastopol: O'Reilly, 2015.

SESHADRI, Shyam; GREEN, Brad. **AngularJS: Up And Running.** Sebastopol: O'Reilly, 2014.

STRONGLOOP. **About.** Disponível em: <http://www.loopback.io>. Acesso em: Novembro de 2015.

STRONGLOOP. **AngularJS JavaScript SDK.** Disponível em: < https://docs.strongloop.com/display/public/LB/AngularJS+JavaScript+SDK>. Acesso em: Novembro de 2015.

FLING, Brian. **Mobile Design and Development.** Sebastopol: O'Reilly, 2009.

FIRTMAN, Maximiliano. **Programming the Mobile Web.** Sebastopol: O'Reilly, 2013.

HOLDENER III, Anthony T. **HTML5 Geolocation.** Sebastopol: O'Reilly, 2011.

IDC. **Smartphone OS Market Share, 2015 Q2.** Disponível em: < http://www.idc.com/prodserv/smartphone-os-market-share.jsp>. Acesso em: Novembro de 2015.

DOGLIO, Fernando. **Pro REST API Development with Node.js.** La Paz: Apress, 2015.

UFRGS. **Concurso Vestibular de 2015.** Disponível em: < http://www.ufrgs.br/coperse/concurso-vestibular/anteriores/2015/concurso-vestibular-2015/EDITALCV2015.pdf>. Acesso em: Novembro de 2015.

GOOGLE PLAY. **UFRGS Mapas.** Disponível em: < https://play.google.com/store/apps/details?id=br.ufrgs.ufrgsmapas&hl=pt_BR>. Acesso em: Dezembro de 2015.

GOOGLE PLAY. **Onde fica? - UFC.** Disponível em: < https://play.google.com/store/apps/details?id=br.ufc.ondefica&hl=pt_BR>. Acesso em: Dezembro de 2015.